TIETO- JA SÄHKÖTEKNIIKAN TIEDEKUNTA

Julius Hekkala

KVANTTITURVALLISTEN ALGORITMIEN TOTEUTTAMINEN KRYPTOGRAFISEEN OHJELMISTOKIRJASTOON

Diplomityö
Tietotekniikan tutkinto-ohjelma
Kesäkuu 2021

Hekkala J. (2021) Kvanttiturvallisten algoritmien toteuttaminen kryptografiseen ohjelmistokirjastoon. Oulun yliopisto, Tietotekniikan tutkinto-ohjelma, 50 s.

TIIVISTELMÄ

Kryptografia on olennainen osa informaatioyhteiskuntaa. Julkisen avaimen salausmenetelmiä käytetään muun muassa kommunikaatioprotokollissa avaintenvaihtoon ja digitaalisiin allekirjoituksiin. Ne perustuvat matemaattisiin ongelmiin, jotka ovat hankalia perinteisille tietokoneille. Kvanttitietokoneen kehittyminen uhkaa kuitenkin niiden turvallisuutta, sillä on olemassa kvanttialgoritmeja, jotka huomattavasti nopeuttavat näiden ongelmien ratkaisuja. Tätä uhkaa vastaan on kehitetty ja kehitetään kvanttiturvallisia algoritmeja, jotka perustuvat ongelmiin, joihin ei kvanttitietokoneillekaan tunneta helppoa ratkaisua.

Tässä työssä integroitiin kopioon Crypto++-ohjelmistokirjastosta kolme varsin tuoretta kvanttiturvallista algoritmia. Niiden suorituskykyä verrattiin keskenään ja muiden algoritmien kanssa. Yleistasolla integroitujen versioiden suorituskyky oli tyydyttävä, joskin hieman heikompi kuin perustana käytetyissä mallitoteutuksissa. Haasteita tässä työssä aiheuttivat muun muassa algoritmien monimutkaisuus, debuggaus ja parametrien järkevä käsittely.

Avainsanat: Kryptografia, kvanttiturvallinen kryptografia, julkisen avaimen salausmenetelmät

Hekkala J. (2021) Implementing Post-Quantum Cryptographic Algorithms. University of Oulu, Degree Programme in Computer Science and Engineering, 50 p.

ABSTRACT

Cryptography is an integral part of the information society. Public key cryptography is used in key exchange in different communication protocols and digital signatures. It is based on mathematical problems that are hard for classic computers. The quantum computer threatens the safety of these algorithms, as there are quantum algorithms that efficiently tackle some of these problems. Post-quantum cryptography has been and is being developed as an answer to this threat.

In this thesis, three quite recent post-quantum algorithms were integrated into a fork of Crypto++, which is an open source cryptographic library. The performance of the algorithms was measured and compared. On a general level, the performance achieved was satisfactory. Big challenges in this work were among others the complexity of the algorithms, debugging process and handling the multitude of parameters of the algorithms.

Keywords: Cryptography, post-quantum cryptography, public key cryptography

SISÄLLYSLUETTELO

ALKULAUSE

Tämä diplomityö tehtiin osana Post-Quantum Cryptography -projektia VTT:llä. Haluasin kiittää VTT:tä ja Kimmo Halusta mahdollisuudesta tehdä diplomityö tästä aiheesta. Haluaisin kiittää professori Juha Röningiä, professori Kimmo Halusta ja Teemu Tokolaa työn ohjaamisesta.

Oulussa 8. kesäkuuta 2021

Julius Hekkala

LYHENTEIDEN JA MERKKIEN SELITYKSET

CCA	Chosen Ciphertext Attack, kryptoanalyysin hyökkäysmalli, jossa hyökkääjä saa tietoonsa selkotekstin haluamilleen salateksteille
CPA	Chosen Plaintext Attack, kryptoanalyysin hyökkäysmalli, jossa hyökkääjä saa tietoonsa haluamiaan selkotekstejä vastaavat salatekstit
DSA	Digital Signature Algorithm, yhdysvaltalainen standardi digitaalisille allekirjoituksille
ECC	Elliptic Curve Cryptography, elliptisten käyrien salausmenetelmät
ECDH	Elliptic Curve Diffie-Hellman, Diffie-Hellman-avaintenvaihdon toteuttaminen elliptisten käyrien avulla
ECDSA	Elliptic Curve Digital Signature Algorithm, digitaalinen allekirjoitusalgoritmi, joka käyttää elliptisiä käyriä
GapSVP	SVP-ongelman tietty versio
GCC	Gnu Compiler Collection, kokoelma kääntäjiä, joilla voi kääntää esim. C- ja C++-kielistä koodia
GCHQ	Government Communications Headquarters, brittiläinen tiedustelu- ja turvallisuusorganisaatio
HTTPS	Hypertext Transfer Protocol Secure, protokolla, jota käytetään tiedon salattuun siirtoon Internetin välityksellä
IoT	Internet of Things, esineiden internet
KDF	Key derivation function, kryptografinen funktio, joka lyhyen syötteen pohjalta generoi salaisia avaimia
KEM	Key Encapsulation Mechanism, tapa jakaa symmetrisiä avaimia
LWE	Learning with Errors, vaikea matemaattinen ongelma, johon moni hilapohjainen salausalgoritmi perustuu
LWR	Learning with Rounding, LWE:n versio, joka perustuu pyöristämiseen satunnaisuuden sijaan
MLWE	Module Learning with Errors, LWE:n modulaarinen versio
NIST	National Institute of Standards and Technology, Yhdysvaltojen standardointiviranomainen
NTT	Number Theoretic Transform, diskreetin Fourier-muunnoksen erityistapaus kokonaisluvuille
RLWE	Ring-Learning with Errors, LWE polynomirenkailla
RSA	Rivest-Shamir-Adleman, tunnetuin julkisen avaimen salausalgoritmi
RSA-KEM	RSA:n käyttö KEM-algoritmina avaintenvaihdossa
SSH	Secure Shell, salausprotokolla, jolla voidaan salata tietoliikennettä
SIVP	Shortest Independent Vector Problem, SVP-ongelman versio
SVP	Shortest Vector Problem, Lyhimmän vektorin ongelma hilassa

TLS	Transport Layer Security, salausprotokolla, jolla salataan Internetin yli menevä tietoliikenne
$a \pmod n$	a modulo n
$syt(a, b)$	a:n ja b:n suurin yhteinen tekijä
$a \equiv r \pmod b$	a on kongruentti r:n kanssa modulo b. a:n ja r:n jakojäännös on sama kun jaetaan luvulla b
G	matemaattinen ryhmä G
$g \in G$	Alkio g kuuluu ryhmään G
$\langle g \rangle$	Ryhmän G alkion g generoima syklinen aliryhmä
$O(n^a)$	Algoritmin aikakompleksisuus on polynominen. Algoritmi suorituu vähintään ajassa n^a, kun algoritmin syöte on n
$\|0\rangle$	Kubitin tila kvanttitietokoneessa
Ψ	Kubittien tilaa kuvaava aaltofunktio
σ_x	Pauli-X -portti, kvanttitietokoneen NOT-portti
$\{\ldots\}$	Joukko
$\mathbb{Z}$	Kokonaislukujen joukko
$\mathbb{Z}_q^{m \times n}$	$m \times n$-kokoisten kokonaislukukertoimisten matriisien joukko (modulo q)
$\mathbb{Z}[X]/(X^n + 1)$	konvoluutiopolynomirengas astetta n
$\mathbb{Z}_q[X]/(X^n + 1)$	konvoluutiopolynomirengas astetta n modulo q

1. JOHDANTO

Ihmisten välisestä kommunikaatiosta yhä suurempi osa tapahtuu verkkoliikenteen välityksellä. Siksi on entistäkin tärkeämpää, että tietoliikenneyhteydet ovat varmasti turvallisia ja liikkuva data oikealla tavalla salattua. Nykyään myös käytännössä meidän kaikkien melkein koko elämä on tallennettuna erilaisiin tietokantoihin. Siksi on oleellista, että tiedot ovat turvassa niiltä, jotka niihin käsiksi pääsessään voisivat käyttää niitä tarkoituksiin, joihin emme ole antaneet suostumusta. Kryptografisia algoritmeja ja ratkaisuja käytetään turvaamaan ihmisten välistä kommunikaatiota ja ihmisten yksityisyyttä sekä datan turvallisuutta. Kryptografiset algorimit perustuvat matemaattisiin ongelmiin, joiden uskotaan olevan niin vaikeita ratkaista, etteivät tietokoneet pysty siihen järkevässä ajassa.

Useat yleisesti käytetyt kryptografiset algoritmit, kuten esimerkiksi julkisen avaimen salausmenetelmiin kuuluvat RSA [1] ja elliptisten käyrien salausmenetelmät [2] perustuvat vaikeasti ratkaistavissa oleviin matemaattisiin ongelmiin. RSA perustuu tekijöihin jakamiseen ja elliptisten käyrien menetelmät diskreetteihin logaritmeihin. Julkisen avaimen salausmenetelmiä käytetään useissa kommunikaatioprotokollissa erityisesti avainten vaihtamiseen. Tunnetuista ja yleisesti käytetyistä protokolllista esimerkiksi Internetin välityksellä tapahtuvan tietoliikenteen salaava TLS (*Transport Layer Security*) [3] ja SSH (*Secure shell*) [4] käyttävät julkisen avaimen salausmenetelmiä avaintenvaihdossa.

Koska tekijöihin jakaminen ja diskreetti logaritmi ovat erittäin hankalia ongelmia perinteiselle tietokoneelle, algoritmit kykenevät vastaamaan laskentatehon kasvuun vain kasvattamalla avainten kokoa. Kvanttitietokoneen tulo kuitenkin uhkaa näitä algoritmeja. Shorin algoritmin [5] avulla kvanttitietokoneet pystyvät tehokkaasti ratkaisemaan tekijöihinjaon, ja samainen algoritmi soveltuu myös diskreettien logaritmien ratkaisemiseen [6]. Mikäli hyökkääjä kykenee saamaan käsiinsä tarpeeksi tehokkaan kvanttitietokoneen, mikä on tulevaisuudessa erittäin todennäköistä, näitä algoritmeja ei enää voi pitää turvallisina. Siksi eri puolilla maailmaa on alettu kiinnittää ongelmaan huomiota, ja tutkijat ovat kehittäneet ja tutkineet kvanttiturvallisia algoritmeja, jotka voisivat korvata julkisen avaimen salausalgoritmeja. Eri maiden standardointiviranomaiset ovat myös alkaneet standardoida kvanttiturvallisia algoritmeja.

Suuri osa erilaisista totetutuksista, joissa kryptografisia algoritmeja käytetään, käyttävät ulkoisia kryptografisia ohjelmistokirjastoja. Mitä algoritmeja näissä toteutuksissa voidaan käyttää, riippuu täysin kirjaston valikoimasta. Tällä hetkellä laajasti käytetyissä kirjastoissa kvanttiturvallisten algoritmien valikoima on hyvin rajallinen. Monet kirjastoista eivät tarjoa ainuttakaan toteutusta uusista kvanttiturvallisista algoritmeista johtuen todennäköisesti pääasiassa niiden nuoresta iästä ja siitä, että ne eivät vielä tällä hetkellä ole välttämättömiä. Esimerkiksi OpenSSL:n [7] perusversiossa ei ole saatavilla uusia kvanttiturvallisia algoritmeja.

Tämän työn tavoitteena oli liittää ohjelmistokirjastoon kvanttiturvallisia algoritmeja, joita kuka tahansa voisi mahdollisimman helposti käyttää omissa ohjelmissaan. Tavoitteena oli analysoida sitten integroinnin onnistumista ja työssä kohdattuja haasteita. Kirjastoksi valittiin Crypto++ [8]. Tässä työssä liitettiin kopioon [9] kirjastosta kolme NIST:n (*National Institute of Standards and Technology*) kvanttiturvallisten algoritmien standardointikilpailun kolmannen kierroksen

algoritmia, joista kaksi on tarkoitettu avaintenvaihtoon (KEM, engl. *Key Encapsulation Mechanism*) [10], ja yksi digitaalinen allekirjoitusalgoritmi.

Toteutetut algoritmit valittiin suorituskyvyn ja standardiksi valitsemisen todennäköisyyden perusteella. Mahdollisimman hyvän suorituskyvyn ja luotettavuuden saavuttamiseksi toteutuksen pohjana käytettiin algoritmien suunnittelijoiden tekemiä mallitoteutuksia. Algoritmien toteutusta ja suorituskykyä arvioitiin, sekä etsittiin mahdollisia ongelmia toteutuksista.

2. JULKISEN AVAIMEN KRYPTOGRAFIA

Nykyisessä informaatioyhteiskunnassa on erityisen tärkeää, että dataliikenne eri kohteiden välillä on tehokkaasti salattua. Erilaisilla salausmenetelmillä pyritään takaamaan se, että salattua dataa eivät pysty lukemaan tai esimerkiksi muokkaamaan muut kuin lähettäjä ja tarkoitettu vastaanottaja. Jotta tämä olisi mahdollista, näillä kahdella osapuolella tulee olla juuri tietyt avaimet, joilla alkuperäisen selkotekstin saa esille. Avaimet eivät myöskään saa päätyä muiden osapuolten käsiin, eli osapuolten tulee onnistua sopimaan yhteisistä avaimista ilman, että niitä suoraan lähetetään toiselle osapuolelle salaamatonta yhteyttä pitkin. Esimerkiksi tässä tilanteessa julkisen avaimen salausmenetelmät ovat tärkeässä asemassa.

Tässä luvussa esitellään julkisen avaimen salausmenetelmien historiaa, niiden perusteita, muutama tärkeä algoritmi ja julkisen avaimen salausmenetelmien perinteisiä käyttökohteita. Julkisen avaimen salausta (sekä muitakin salausmenetelmiä) on eritelty tarkemmin esimerkiksi kirjassa [11].

2.1. Julkisen avaimen kryptografian historiaa

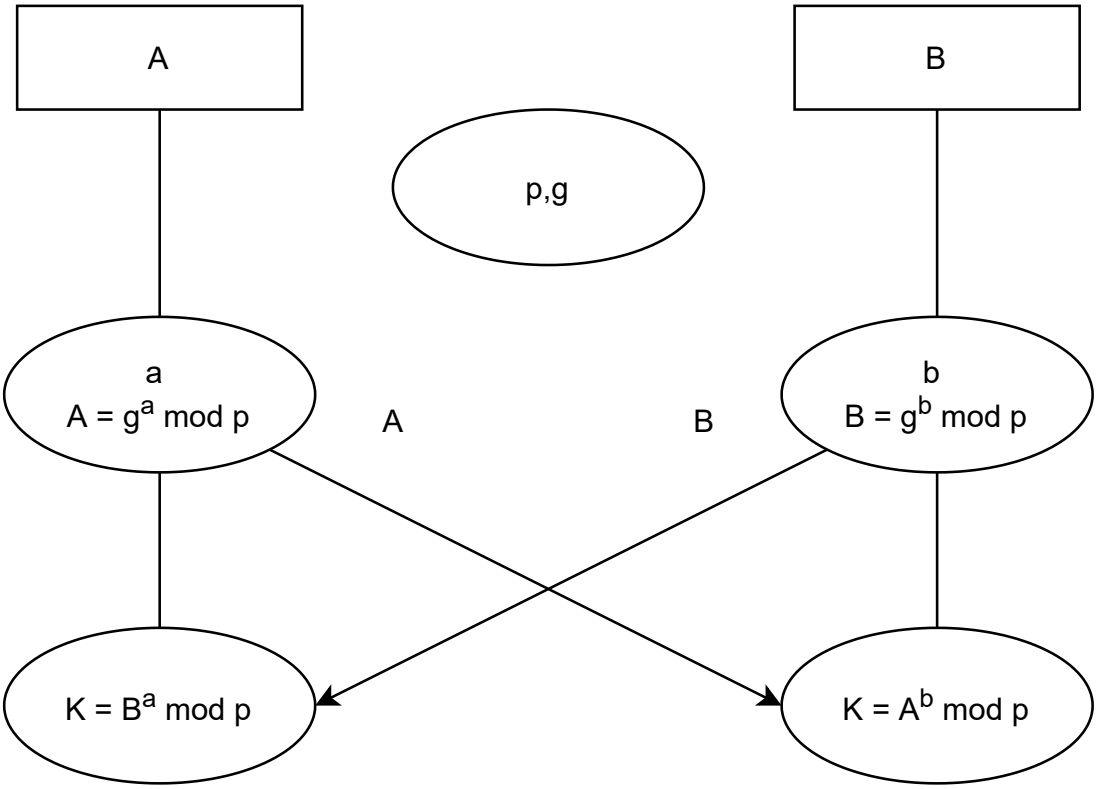
Ajatus julkisen avaimen kryptografiasta tai asymmetrisesta kryptografiasta on varsin nuori. Julkisen avaimen kryptografiaa esittelivät ensimmäisen kerran Diffie ja Hellman vuonna 1976 [12]. Yksinkertaistettu kuvaus Diffie-Hellman-avaintenvaihdosta on esitetty kuvassa 1. Yksinkertaisimmillaan Diffie-Hellman-avaintenvaihdossa aivan aluksi osapuolet A ja B sopivat yhteisistä luvuista p ja g. p on suuri alkuluku ja g sen primitiivinen juuri. A:lla ja B:llä on salaiset potenssit a ja b, ja A laskee $A = g^a \pmod p$ ja B laskee $B = g^b \pmod p$. A lähettää B:lle luvun A ja B lähettää A:lle luvun B. A laskee $K = B^a \pmod p$ ja B laskee $K = A^b \pmod p$. Näin lopulta sekä A:lla että B:llä on sama yhteinen salaisuus K.

Yksi tunnetuimmista julkisen avaimen algoritmeista, RSA, esiteltiin hieman myöhemmin [1]. RSA:ta on tarkemmin esitelty tämän luvun kohdassa 2.2.1. RSA:ta vastaava salausjärjestelmä oli kehitetty jo joitakin vuosia aiemmin brittiläisessä tiedustelu- ja turvallisuuspalvelu GCHQ:ssa (engl. *Government Communications Headquarters*), mutta tämä julkistettiin vasta vuonna 1997.

2.2. Julkisen avaimen kryptografian perusteet

Perinteisesti salausmenetelmät perustuvat siihen, että kaikilla osapuolilla on tiedossa sama avain, jolla teksti on salattu. Salatun tekstin avaaminen tapahtuu samalla avaimella. Tällaisia salausjärjestelmiä kutsutaan symmetrisiksi salausjärjestelmiksi. Julkisen avaimen salausjärjestelmissä jokaisella on sekä julkinen että yksityinen avain. Niitä kutsutaan myös epäsymmetrisiksi salausjärjestelmiksi. Kuka tahansa voi ottaa selville kenen tahansa julkisen avaimen ja salata tekstejä tällä avaimella. Salauksen avaaminen onnistuu ainoastaan julkista avainta vastaavalla yksityisellä avaimella.

Julkisen avaimen kryptografia on olennainen osa nykyistä informaatioyhteiskuntaa. Esimerkiksi TLS-protokollan [3] avaintenvaihto perustuu julkisen avaimen salausmenetelmiin. TLS on protokolla, jota käytetään erityisesti HTTPS-protokollassa

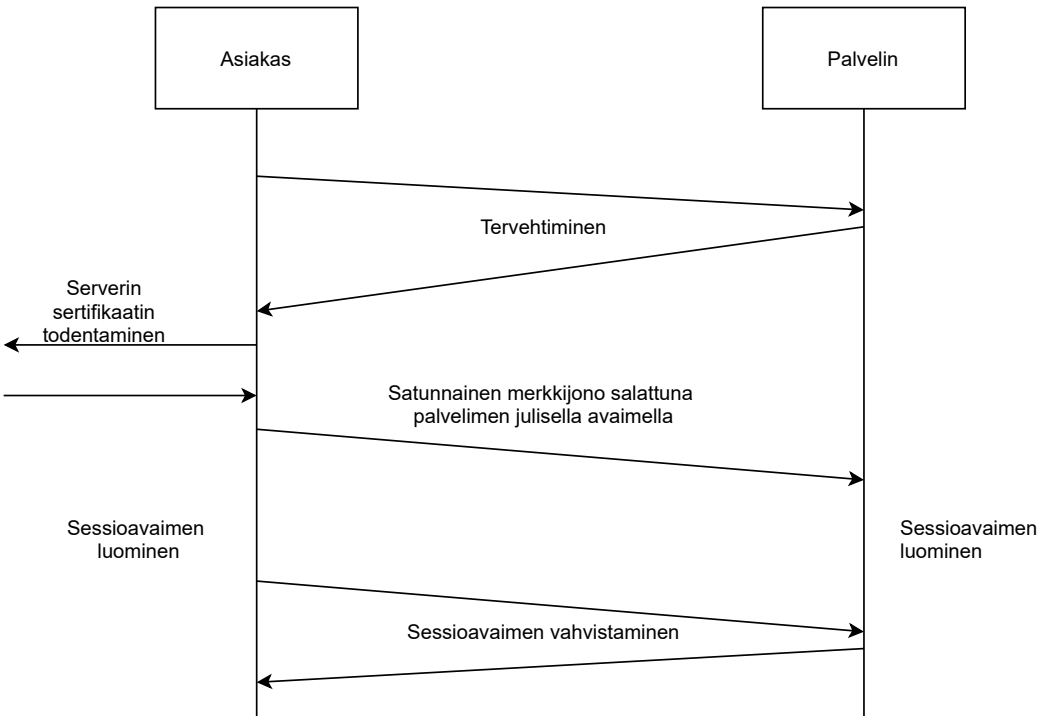


Kuva 1. A ja B muodostavat yhteisen salaisuuden K Diffie-Hellman-avaintenvaihdon välityksellä.

(engl. *Hypertext Transfer Protocol Secure*) [13] Internetin välityksellä kulkevan tiedon salaamisen, joten se on keskeisesti läsnä kaikkien Internetiä käyttävien tahojen elämässä.

TLS:n kättelyksi kutsutaan prosessia, jossa palvelin ja asiakasprosessi muun muassa autentikoivat toisensa ja sopivat sessioavaimet. TLS:n kättelyä (TLS:n versiolle 1.2) on yksinkertaistettuna kuvattu kuvassa 2. Tervehtimisvaiheessa sekä asiakas että palvelin lähettävät toisilleen satunnaiset merkkijonot, joita yhdessä asiakkaan myöhemmin lähettämän salatun kolmannen merkkijonon kanssa käytetään sessioavainten luomiseen. Tervehtimisvaiheessa asiakas ja palvelin myös sopivat siitä, mitä TLS:n versiota ne käyttävät ja mitä salausalgoritmia käytetään kättelyvaiheessa sekä session aikana. Sessioavaimen vahvistamisvaiheessa molemmat lähettävät toisilleen sessioavaimella salatun viestin tarkistaakseen, että kättelyprotokolla suoriutui onnistuneesti.

Julkisen avaimen kryptografia perustuu matemaattisiin ongelmiin, joita on helppo ratkaista yhteen suuntaan, mutta todella hankalaa ratkaista toiseen suuntaan (engl. *trapdoor function*). Käytännössä tämä tarkoittaa, että ongelman ratkaisevan algoritmin kompleksisuus on superpolynominen, eli ongelmalle ei ole olemassa algoritmia, joka ratkaisisi sen polynomiajassa. Polynomiajassa suoriutuvia algoritmeja pidetään riittävän tehokkaina, että niitä realistisesti voidaan käyttää. Ongelmaa, jolla ei ole polynomiajassa suoriutuvaa ratkaisua, kutsutaan NP-hankalaksi. [11]



Kuva 2. Yksinkertaistettu kuvaus TLS 1.2-protokollan kättelystä.

Tunnetuimmat julkisen avaimen salausmenetelmissä hyödynnetyt matemaattiset ongelmat ovat tekijöihinjako ja diskreetit logaritmit. Esimerkiksi RSA [1] perustuu tekijöihinjakoon ja elliptisen käyrän menetelmät [2] diskreetteihin logaritmeihin.

Julkisen avaimen salausmenetelmissä on kahdenlaisia avaimia, julkisia ja yksityisiä avaimia. Yksinkertaistettuna näistä julkisen avaimen voi ottaa selville kuka tahansa ja salata sillä viestejä, jotka voi avata ainoastaan vastaavalla yksityisellä avaimella. Digitaalisissa allekirjoituksissa taas avaimia käytetään toiseen suuntaan. Dokumentin allekirjoittaja käyttää yksityistä avaintaan allekirjoitukseen ja kuka tahansa voi tarkistaa allekirjoituksen oikeellisuuden allekirjoittajan julkisella avaimella.

2.2.1. Tekijöihin jakaminen ja RSA

RSA on yksi merkittävimmistä julkisen avaimen salausmenetelmistä ja se esiteltiin ensimmäisen kerran vuonna 1978 [1]. RSA perustuu siihen, että tietyissä tilanteissa tekijöihin jakaminen on erittäin vaikeaa. Kun $n = p * q$ ja p ja q ovat suurikokoisia alkulukuja, luvun n jakaminen tekijöihin lukujen p ja q löytämiseksi on todella vaikeaa perinteiselle tietokoneelle. Tätä ominaisuutta hyväksikäyttäen voi suunnitella varsin yksinkertaisia, mutta tehokkaita salausmenetelmiä.

RSA:ssa julkinen avain koostuu kahdesta positiivisesta kokonaisluvusta e ja n, ja yksityinen avain positiivisista kokonaisluvuista d ja n. Lukujen valinta RSA:ssa tapahtuu seuraavalla tavalla:

- Valitaan kaksi satunnaista suurta alkulukua p ja q ja niiden avulla lasketaan $n = p * q$.

- Sen jälkeen valitaan luvuksi d suuri kokonaisluku, joka täyttää ominaisuuden $syt(d, (p-1)*(q-1)) = 1$ (syt = suurin yhteinen tekijä).
- Lopuksi e lasketaan siten että kaava $e * d \equiv 1 \pmod{(p-1)*(q-1)}$ pätee.

Kun tarvittavat kokonaisluvut on saatu määritettyä, RSA:ssa viesti M salataan seuraavasti salausalgoritmilla E:

$$C \equiv E(M) \equiv M^e \pmod{n}, \tag{1}$$

jolloin C on syntyvä salateksti, ja e ja n aiemmin kuvatulla tavalla valitut positiiviset kokonaisluvut. Salatekstin C salauksen purkaminen taas tapahtuu seuraavasti salausalgoritmilla D:

$$M \equiv D(C) \equiv C^d \pmod{n}, \tag{2}$$

jolloin M on lopputuloksena saatava salattuna ollut selkoteksti.

2.2.2. Diskreetit logaritmit ja elliptisten käyrien salausalgoritmit

Useat salausjärjestelmät, esimerkiksi myös aiemmin mainittu Diffie-Hellman [12], perustuvat diskreetteihin logaritmeihin. Diskreetin logaritmin ongelma on lyhyesti seuraavanlainen:

Olkoon G matemaattinen ryhmä ja yksi sen alkioista on $g \in G$. Olkoon $\langle g \rangle$ alkion g generoima syklinen aliryhmä, eli kaikki sen jäsenet ovat luvun g kokonaislukupotensseja. Tällöin diskreetin logaritmin ongelma on seuraava: jos on olemassa kyseiseen sykliseen aliryhmään kuuluva kokonaisluku $a \in \langle g \rangle$, mikä on kokonaisluku x niin että $g^x = a$. [14]

Elliptisten käyrien salausalgoritmien (ECC, engl. *Elliptic Curve Cryptography*) historia alkaa 1980-luvun lopulta [15][16]. Elliptisen käyrän määrittelee yhtälö

$$y^2 = x^3 + ax + b, \tag{3}$$

jossa a ja b kuuluvat kuntaan K. Kunta on matemaattinen joukko, jossa on määritelty yhteen- ja kertolasku, ja tavallisia laskusääntöjä noudattamalla operaatioiden tulokset kuuluvat myös kuntaan. Elliptisen käyrän salausalgoritmit toimivat vain tietynlaisissa kunnissa.

Verrattuna esimerkiksi RSA:han, elliptisten käyrien algoritmeissa on yleensä pienemmät avainkoot, joten sama turvallisuustaso pystytään saavuttamaan huomattavasti edullisemmin. Taulukossa 1 on eritelty RSA:n ja elliptisten käyrien salausmenetelmien avainten suositeltua kokoeroa eri turvallisuustasoilla bitteinä [17]. Varsinkin korkeammilla turvallisuustasoilla kokoero avainten välillä on suuri. 80:n bitin turvallisuustasoa ei enää pidetä turvallisena, ja 112:n bitin turvallisuustaso onkin alhaisin turvallisuustaso, jota NIST suosittelee käytettävän.

Elliptisten käyrien salausalgoritmeja on suunniteltu ja standardoitu useita erilaisia. Yksi tunnetuimmista tavoista käyttää elliptisiä käyriä on liittää ne Diffie-Hellman -avaintenvaihtoon (ECDH, engl. *Ellipctic Curve Diffie-Hellman*), kuten esimerkiksi Curve25519 [18], joka on yksi nopeimmista elliptisistä käyristä.

Taulukko 1. NIST:n suosittelemat RSA:n ja ECC:n avainten koot eri turvallisuustasoilla.

Turvallisuustaso	Avainten koko	
bitteinä	RSA	ECC
(80)	(1024)	(160)
112	2048	224
128	3072	256
192	7680	384
256	15360	512

Curve25519-algoritmia käytetään avaintenvaihtoon lukuisissa eri sovelluksissa, kuten esimerkiksi WhatsAppissa[1] ja Signalissa[2]. Elliptisiä käyriä käytetään myös esimerkiksi digitaalisiin allekirjoituksiin (ECDSA, engl. *Elliptic Curve Digital Signature Algorithm*) [19].

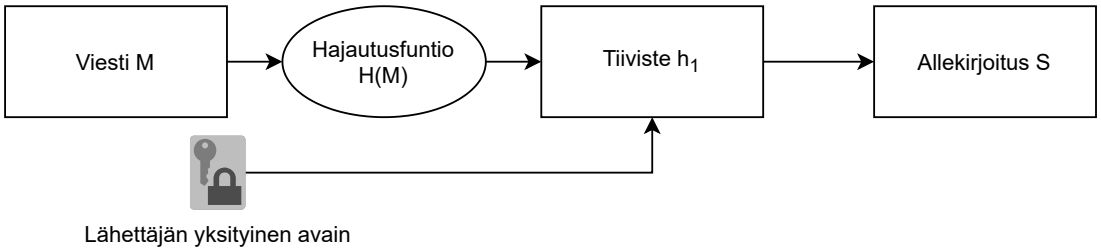
2.3. Digitaaliset allekirjoitukset

Digitaalisten allekirjoitusten avulla voidaan varmistaa, että dokumentin kirjoittaja on juuri se, kuka hän sanoo olevansa ja että dokumenttia ei ole muokattu allekirjoituksen tekemisen jälkeen. Digitaalisten allekirjoitusten algorimit ovat yleensä julkisen avaimen salausmenetelmiä. Allekirjoittaja käyttää yksityistä avaintaan allekirjoittaakseen dokumentin, ja muut voivat allekirjoittajan julkista avainta käyttäen varmistaa allekirjoituksen aitouden. Koska muut eivät tiedä allekirjoittajan yksityistä avainta, digitaalista allekirjoitusta on hyvin vaikeaa väärentää.

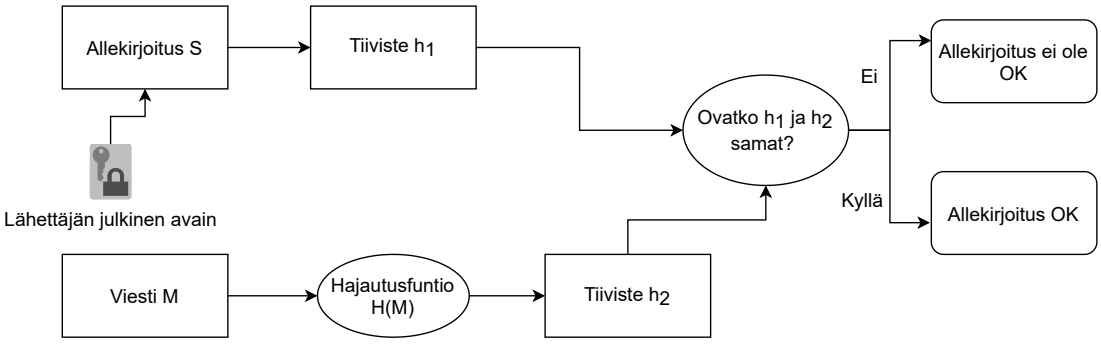
Tyypillisesti digitaalisissa allekirjoituksissa allekirjoitettavasta viestistä lasketaan hajautusalgoritmilla (engl. *hash function*) tiiviste, ja viesti salataan lähettäjän yksityisellä avaimella. Tämä allekirjoitus, sekä alkuperäinen viesti ja lähettäjän julkinen avain pakataan kokonaisuudeksi ja salataan vastaanottajan julkisella avaimella. Vastaanottaja kykenee siten avaamaan viestin ja varmistamaan lähettäjän julkisen avaimen avulla että viesti todellakin on tämän lähettämä eikä sitä ole muokattu. [20]

Kuvassa 3 esitetään digitaalisen allekirjoituksen luominen viestille M ja kuvassa 4 esitetään digitaalisen allekirjoituksen varmistaminen samaiselle viestille yksinkertaistettuna. NIST on julkaissut standardin [21], jossa määritellään RSA-algoritmiin [1] perustuvat digitaaliset allekirjoitukset, elliptisten käyrien digitaaliset allekirjoitukset (ECDSA) [19] ja digitaalinen allekirjoitusalgoritmi DSA (engl. *Digital Signature Algorithm*).

[1] https://www.whatsapp.com
[2] https://signal.org/



Kuva 3. Digitaalisen allekirjoituksen luominen viestille M.



Kuva 4. Digitaalisen allekirjoituksen varmistaminen.

3. KVANTTITURVALLINEN KRYPTOGRAFIA

Julkisen avaimen salausalgoritmeista suuri osa perustuu sellaisiin matemaattisiin ongelmiin, jotka ovat vaikeita perinteiselle tietokoneelle, mutta joille on olemassa kvanttilaskentaan perustuvia ratkaisuja, joilla ongelmat ratkevat huomattavasti helpommin. Esimerkiksi Shorin algoritmi [5] mahdollistaa suurten kokonaislukujen alkulukutekijöiden selvittämisen kvanttilaskennan avulla aiempaa huomattavasti tehokkaammin.

Mikäli hyökkääjillä on tulevaisuudessa saatavilla tarpeeksi tehokkaita kvanttitietokoneita, aiheuttaa se erittäin suuren uhan erilaisten järjestelmien turvallisuudelle. Jos tai kun perinteisiä julkisen avaimen algoritmeja ei voi enää pitää turvallisina, täytyy järjestelmissä niiden toiminnallisuus jotenkin korvata muilla algoritmeilla. Tässä tilanteessa kuvaan astuvat nk. kvanttiturvalliset algoritmit (engl. *post-quantum cryptography*), joiden tarkoitus on korvata julkisen avaimen salausmenetelmiä. Kvanttiturvalliset algoritmit perustuvat ongelmiin, joihin ei tunneta kvanttilaskentaratkaisuja eikä myöskään klassisella tietokoneella suoritettavia ratkaisualgoritmeja, jotka suoriutuisivat polynomisessa ajassa eli ratkaisun kompleksisuus olisi $O(n^a)$, missä a on vakio.

3.1. Kvanttitietokoneen toiminnan perusteet

Kvanttitietokoneet hyödyntävät laskennassa kvanttitason ilmiöitä, kuten esimerkiksi kubittien superpositioita. Kvanttitietokoneen toimintaa tarkemmin on esitelty esimerkiksi papereissa [22] ja [23]. Kvanttitietokoneet mahdollistavat kvanttilaskennan tekemisen koneellisesti.

Perinteisessä tietokoneessa kaikilla biteillä on tietty selvä tila, joko 0 tai 1. Kvanttitietokoneen yksikköjä kutsutaan kubiteiksi. Jos kvanttitietokoneessa on n kubittia, niin sillä on 2^n keskenään ortogonaalista mahdollista kvanttitilaa. Kubitin mahdolliset tilat ovat $\{|0\rangle, |1\rangle\}$. [23]

Kubittien tilaa voi kuvata aaltofunktiolla

$$\Psi = a|01100\ldots\rangle + b|10001\ldots\rangle + \ldots, \tag{4}$$

missä $a, b, \ldots$ ovat kompleksisia kertoimia ja todennäköisyys sille että kubitit ovat tilassa $01100\ldots$ on $|a|^2$ ja niin edelleen vastaavasti muille kertoimille. Kvanttitietokoneen kubitit ovat kaikissa mahdollisissa tiloissa niin kauan kun niitä ei mitata, mitatessa ne ovat edellä mainitulla todennäköisyydellä tietyssä tilassa. Kvanttilaskenta perustuu kvanttiporttien käyttöön. Niiden avulla kubittien tilaa muokataan, jotta saavutetaan haluttu lopputulos. Kvanttiporttien avulla voidaan rakentaa kvanttipiirejä. [22] Esimerkiksi *Pauli-X* -portti on matriisiesityksenä seuraavanlainen:

$$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{5}$$

.

Pauli-X -portti muuttaa tilassa $|0\rangle$ olevan kubitin tilaan $|1\rangle$ ja päinvastoin. Kyseinen kvanttiportti vastaa siis klassista NOT-porttia.

3.2. Kvanttitietokoneen aiheuttama uhka perinteisille julkisen avaimen algoritmeille

Kvanttilaskenta mahdollistaa useiden perinteiselle tietokoneelle käytännössä mahdottomien ongelmien ratkaisemisen. Kvanttitietokoneet kykenevät ratkaisemaan tiettyjä ongelmia merkittävästi perinteisiä tietokoneita nopeammin. Esimerkiksi Groverin algoritmi [24] on etsintäalgoritmi, joka nopeuttaa väsytyshyökkäyksen (*brute-force* -hyökkäys) tekemistä salausavaimen löytämiseksi kryptografisia algoritmeja kohtaan. Tämän takia symmetristen salausjärjestelmien avainkoot pystytään selvittämään nopeammin kvanttitietokoneella. Käytännössä tämä tarkoittaa, että symmetrisissä salausjärjestelmissä avainten koot pitää tuplata pitääkseen turvallisuustason entisenä.

Asymmetrisiä eli julkisen avaimen salausjärjestelmiä kohtaan kvanttitietokoneen vaikutus on huomattavasti merkittävämpi. Shorin algoritmi [5] mahdollistaa erittäin suurien kokonaislukujen alkulukutekijöiden laskemisen polynomiajassa eli tehokkaasti. Kun esimerkiksi RSA perustuu juurikin tekijöihinjakoon, ei sitä voi pitää turvallisena maailmassa, jossa on pääsy tarpeeksi tehokkaisiin kvanttitietokoneisiin. Shorin algoritmia voi käyttää myös elliptisten käyrien salausmenetelmien rikkomiseen tehokkasti [6].

Vielä tämän hetken saatavilla olevilla kvanttitietokoneilla eivät julkisen avaimen salausmenetelmät ole vaarassa, mutta mikäli tarpeeksi tehokkaita kvanttitietokoneita, joissa on tarpeeksi kubitteja, pystytään kehittämään, ne pystytään purkamaan huomattavasti nopeammin kuin nykyisin. Esimerkiksi 1024-bittisen RSA-salauksen purkaminen vaatii noin 2000 kubittia, tai 160 bitin elliptisen käyrän salauksen purkaminen vaatii noin 1000 kubittia [6]. Esimerkiksi Googlella on käytössään 53 kubitin kvanttitietokone [25]. Mikään ei myöskään takaa, etteikö kvanttitietokoneille ja kvanttilaskentaan voisi löytyä uusia, vielä tuntemattomia tapoja ratkaista erilaisia ongelmia, joihin eri salausjärjestelmät perustuvat. On myös epävarmaa, tuleeko koskaan olemaan niin tehokkaita kvanttitietokoneita, jotka kykenisivät tehokkaasti murtamaan julkisen avaimen salausmenetelmiä. Tähän mahdollisuuteen ei kuitenkaan kannata tuudittautua, vaan on järkevää joka tapauksessa varautua kvanttitietokoneiden muodostamaan uhkaan.

3.3. KEM-algoritmit

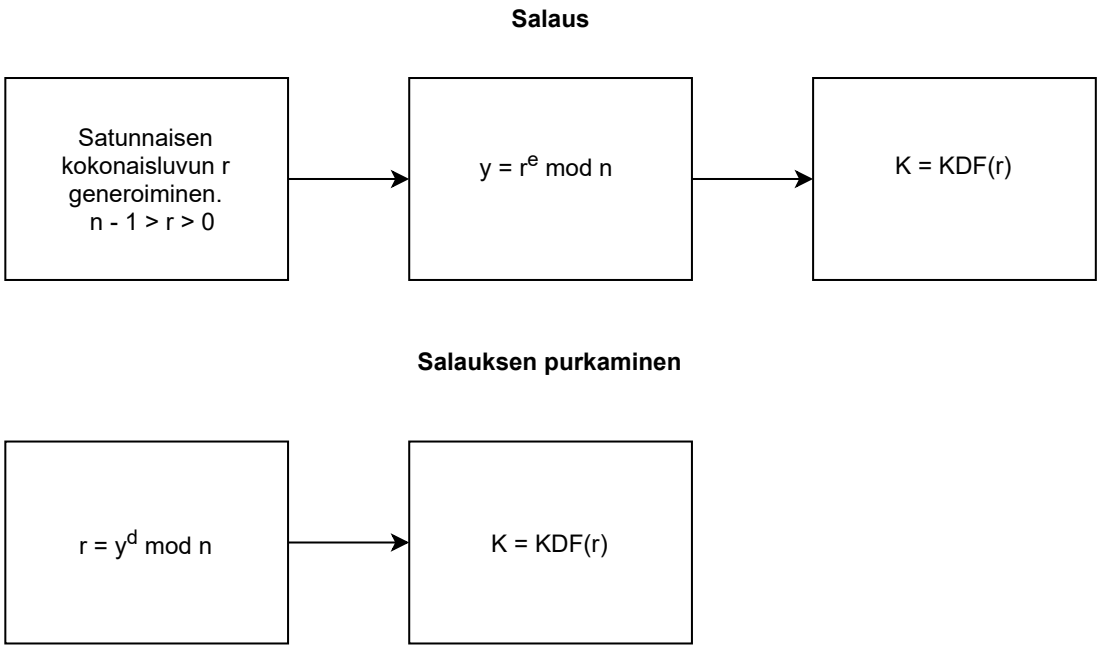
Monissa protokollissa käytetään julkisen avaimen algoritmeja siihen, että osapuolet sopivat kommunikaatioon käytettävän symmetrisen salausalgoritmin avaimista. KEM-algoritmit (engl. *key encapsulation mechanisms*, vapaasti suomennettuna avainten kapselointimekanismit) ovat tehokas tapa suorittaa avaintenvaihto symmetristä salausmenetelmää varten. Tässä kappaleessa esitetään lyhyesti niiden toimintaperiaate yleistasolla. Tällä hetkellä KEM-algoritmien käyttö protokollissa avaintenvaihtoon ei

ole yleistä. NIST:n standardointiprosessissa KEM-algoritmit ovat kuitenkin toinen algoritmien alaluokka.

Abstraktilla tasolla KEM-algoritmit koostuvat kolmesta osasta [10]:

- Avainten generointialgoritmi, jonka tulosteena ovat yksityinen avain SK ja julkinen avain PK.
- Salausalgoritmi, joka ottaa syötteenä julkisen avaimen PK ja tulostaa avain-salateksti-parin (K, C_0).
- Salauksen purkamisalgoritmi, joka ottaa syötteenä yksityisen avaimen SK ja salatekstin C_0 ja tulostaa avaimen K.

Esimerkiksi RSA-algoritmia voidaan käyttää KEM-tyyliin. Kuvassa 5 on esitetty yksinkertaistettuna RSA-KEM-algoritmin toiminta. [10] Avainten generointiosuus RSA-KEM-algoritmissa on täysin samanlainen kuin RSA:ssa yleensäkin, ja on esitelty luvussa 2.2.1. Sen seurauksena kaikilla on julkinen avain, joka koostuu moduluksesta n ja salauseksponentista e sekä yksityinen avain, joka koostuu salauksen purkueksponentista d.



Kuva 5. Yksinkertaistettu kuvaus siitä, miten RSA-KEM-algoritmilla salaus ja salauksen purkaminen toimivat. Lopputuloksina molemmilla osapuolilla on yhteinen avain K. KDF on funktio, joka syötteen perusteella generoi salaisen avaimen (engl. *Key derivation function*).

3.4. Kvanttiturvalliset algoritmit

Koska kvanttitietokoneet asettavat suuren uhan julkisen avaimen salausmenetelmille, nousee esille kysymys siitä, miten niiden käyttö voitaisiin korvata sellaisilla

algoritmeilla, jotka eivät hajoa käsiin tarpeeksi tehokkaan kvanttitietokoneen käsissä. Kvanttiturvallinen kryptografia (*post-quantum cryptography*) [26] käsittää eri kryptografisia algoritmeja, joiden on tarkoitus olla vaikeita myös kvanttitietokoneelle. Niillä on tarkoitus korvata vanhoja ei-kvanttiturvallisiin ongelmiin perustuvia algoritmeja. Kvanttiturvallisiksi algoritmeiksi kutsutaan algoritmeja, jotka perustuvat erilasiin perinteisen matematiikan ongelmiin ja joita suoritetaan klassisella tietokoneella, mutta jotka ovat turvallisia sekä klassista että kvanttitietokonetta vastaan. Kvanttikryptografia (*quantum cryptography*) taas tarkoittaa kvanttilaskentaa hyödyntäviä kryptografisia algoritmeja, ja ne ovat asia erikseen, eikä niitä tässä työssä käsitellä.

Yhdysvaltojen standardointiviranomainen NIST (*National Institute of Standards and Techonology*) aloitti vuonna 2017 kvanttiturvallisten algoritmien standardointikilpailun [27]. Kilpailussa on tarkoitus standardoida kvanttiturvallisia algoritmeja avaintenvaihtoon (KEM-algoritmit) sekä digitaalisiin allekirjoituksiin. Tämän tekstin kirjoittamishetkellä kilpailu on kolmannella ja viimeisellä kierroksella. Taulukossa 2 on esitetty finalistialgoritmit sekä avaintenvaihtoon (KEM) että digitaalisiin allekirjoituksiin tarkoitetuista algoritmeista. NIST ei ole myöskään ainoa standardointiviranomainen, joka standardoi kvanttiturvallisia algoritmeja, mutta sen päätökset ovat erittäin merkittäviä suomalaiselle ja eurooppalaiselle teknologiaekosysteemille.

Taulukko 2. NIST:n kvanttiturvallisten algoritmien standardointikilpailun kolmannen kierroksen finalistialgoritmit.

Algoritmi	Mihin algoritmi perustuu	Algoritmin käyttökohde
Classic McEliece	Goppa-koodit	KEM
CRYSTALS-Kyber	Hilapohjaiset algoritmit	KEM
NTRU	Hilapohjaiset algoritmit	KEM
SABER	Hilapohjaiset algoritmit	KEM
CRYSTALS-Dilithium	Hilapohjaiset algoritmit	Digitaaliset allekirjoitukset
FALCON	Hilapohjaiset algoritmit	Digitaaliset allekirjoitukset
Rainbow	Usean muuttujan polynomit	Digitaaliset allekirjoitukset

Julkisen avaimen algoritmien korvaaminen eri protokollissa ei ole ongelmatonta. Crockett ja muut kuvailevat yrityksiään vaihtaa avaintenvaihto toimimaan kvanttiturvallisilla algoritmilla tai hybridimallilla (käyttämällä sekä perinteisiä että kvanttiturvallisia algoritmeja) SSH:ssa ja eri TLS:n versioissa [28]. He luettelevat tuloksia osalle NIST:n standardointiprosessin toiselle kierrokselle selvinneistä algoritmeista. Ongelmia aiheuttaa muun muassa osan kvanttiturvallisista algoritmeista suuret avaintenkoot. Samoin SSH:n rajoitteet viestin koossa aiheuttavat rajoituksia, sillä osa kvanttiturvallisista algoritmeista tuottaa varsin suuria salatekstejä.

3.5. Hilat ja hilaongelmat

Tässä kappaleessa esitellään lyhyesti hilan käsite sekä yleisellä tasolla muutama hiloihin liittyvä matemaattinen ongelma, joita voidaan käyttää hyödyksi kryptografiassa. Hiloista tarkemmin on kerrottu englanniksi esimerkiksi kirjassa [29]. Suomeksi hiloista ja niiden suhteesta kryptografiaan voi lukea esimerkiksi Olaussenin pro gradusta [30].

Hila on joukko pisteitä, jonka virittää joukko vektoreita, eli sen kanta. Hila muodostuu, kun kannan muodostavia vektoreita kerrotaan kokonaislukukertoimilla. Hilan L matemaattinen esitys on siis

$$L = \{a_1 v_1 + a_2 v_2 + \ldots + a_n v_n\}, \tag{6}$$

jossa $a_1, a_2, \ldots a_n$ kuuluvat kokonaislukujen $\mathbb{Z}$ joukkoon ja $v_1, v_2, \ldots v_n$ ovat hilan kannan muodostavat vektorit. Hilan kannan vektorien määrä on hilan aste. [30]

Yksi keskeisimmistä hilaongelmista on lyhimmän vektorin ongelma (SVP, engl. *Shortest vector problem*). Lyhimmän vektorin ongelman ratkaisee nollasta eroava vektori v, joka on pituudeltaan lyhyin hilan kantavektoreista. Tietyillä ehdoilla SVP on NP-hankala ongelma, eli siihen ei ole polynomiajassa suoriutuvaa ratkaisua. [30]

SVP:stä on esimerkiksi versiot GapSVP ja SIVP (engl. *Shortest independent vector problem*). Näiden ongelmien perusteella Regev [31] on muodostanut LWE-ongelman (engl. *Learning with errors*), johon pohjautuu usea NIST:n standardointikilpailun algoritmeista. LWE-ongelman mukaan jakaumaa $(A, As + e)$ on vaikea erottaa tasajakaumasta, kun A on satunnainen matriisi joukossa $\mathbb{Z}_q^{m \times n}$, s on tasaisesti jakautunut satunnaisvektori joukossa $\mathbb{Z}_q^n$ ja e on pienikertoiminen satunnaisvektori [32]. Tästä ongelmasta on myöhemmin johdettu erilaisia variantteja, kuten Ring-LWE (RLWE) ja Module-LWE (MLWE), johon esimerkiksi Kyber [32] ja Dilithium [33] perustuvat.

LWE-ongelmasta on kehitetty LWR-muoto (engl. *Learning with rounding*). LWR-ongelma eroaa siinä LWE-ongelmasta, että se on deterministinen. LWR-ongelman virhe hankitaan satunnaisuuden sijaan kertoimia pyöristämällä. Esimerkiksi SABER perustuu LWR-ongelman modulaariseen versioon. [34]

3.5.1. Hiloihin perustuvat kvanttiturvalliset algoritmit

Hiloihin perustuvat algoritmit ovat tällä hetkellä nousemassa yhdeksi todennäköisimmistä korvaajista vanhoille salausmenetelmille. NIST:n kvanttiturvallisten algoritmien standardointikilpailun kolmannen kierroksen finalistinalgoritmeista suurin osa on hiloihin perustuvia algoritmeja, kuten on nähtävissä taulukosta 2. Vaikka hila-algoritmit ovat varsin nuoria, niitä pidetään turvallisina ja ne ovat suhteellisen tehokkaita, eikä hila-algoritmeissa yleensä avainten tai salatekstin koko ole jättimäisen suuri.

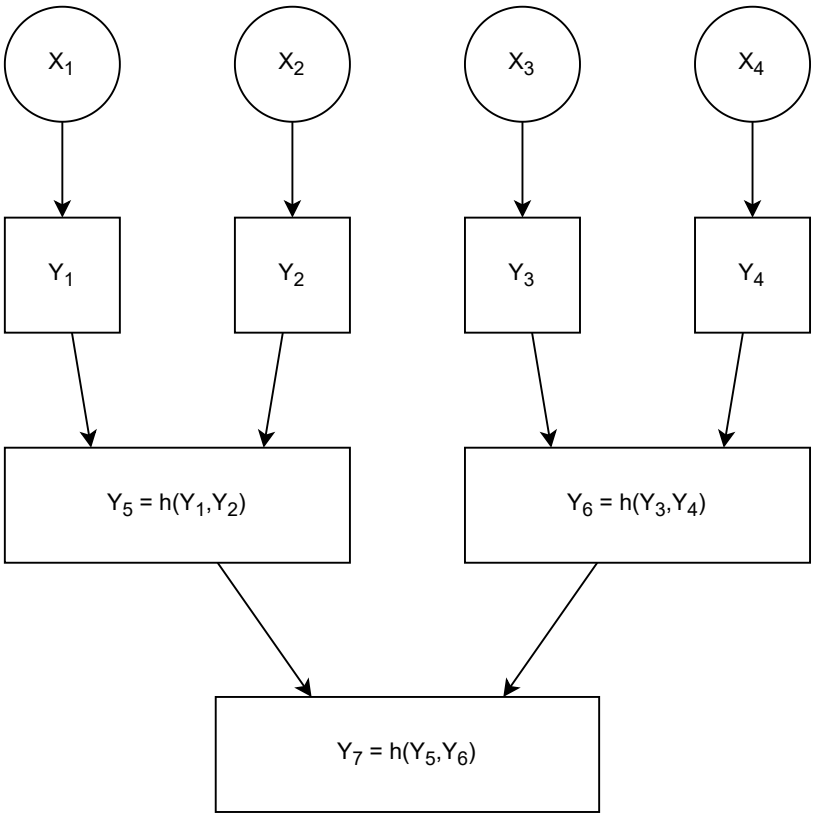
NIST:n standardointikilpailun finalisteista MLWE-ongelmaan perustuvat CRYSTALS-Kyber [32], CRYSTALS-Dilithium [33]. NTRU [35] perustuu siihen että lyhyimmän vektorin löytäminen hilassa (SVP) on vaikeaa. SABER [34] perustuu LWR-ongelmaan [36], ja FALCON [37] perustuu NTRU-hiloihin.

3.6. Muut kvanttiturvalliset algoritmit

Hiloihin perustuvat algoritmit eivät suinkaan ole ainoita julkisen avaimen salausalgoritmeja, jotka kestävät kvanttitietokoneella tapahtuvan hyökkäyksen. Myös muihin matemaattisiin ongelmiin perustuvia kvanttiturvallisia algoritmeja on olemassa. McEliece-järjestelmä [38] perustuu koodausteoriaan. Alkuperäisen algoritmin McEliece esitteli jo vuonna 1978. McEliecen algoritmi perustuu virheenkorjauskoodeihin. Sen avaimet ovat todella suurikokoisia, mutta sen tuottamat salatekstit taas ovat todella pieniä [39]. Nykyisessä versiossa on tiettyjä parannuksia verrattuna alkuperäisiin, mutta perusidea on samanlainen. Koska McEliecen salausjärjestelmää on tutkittu niin pitkään eivätkä hyökkäykset ole parantuneet merkittävästi, sitä pidetään todella luotettavana algoritmina, joka soveltuu tietynlaisiin tilanteisiin [40].

Usean muuttujan polynomeihin perustuviin algoritmeihin kuuluu muun muassa taulukon 2 Rainbow [41]. Usean muuttujan polynomien salausmenetelmät perustuvat siihen, että yleensä neliöllisten polynomien ratkaiseminen äärellisessä kunnassa on hankalaa. Menetelmiä vastaan tunnetaan useita erilaisia hyökkäyksiä, ja algoritmien historian saatossa moni usean muuttujan polynomialgoritmeista onkin osoitettu turvattomiksi [42]. Usean muuttujan polynomien salausmenetelmien suuri heikkous on erittäin suuri avainten koko. Esimerkiksi Rainbowin julkisen avaimen koko on turvallisuustasolla 3 861,4 kilotavua ja yksityisen avaimen koko 611,3 kilotavua. Allekirjoituksen koko taas on huomattavasti pienempi, 1312 bittiä (164 tavua). Rainbow on kuitenkin tehokas allekirjoituksissa ja allekirjoitusten varmistamisessa, mutta suuren avainten koon takia ei sovellu yleiskäyttöiseksi allekirjoitusalgoritmiksi [40].

On lisäksi olemassa muun muassa hash-funktiohin perustuvia salausalgoritmeja. Hash-funktiot perustuvat siihen, että niitä on käytännössä mahdoton laskea käänteiseen suuntaan. Hash-funktioiden käytössä iso haittapuoli on se, että salaista avainta voi käyttää vain yhden allekirjoituksen tekemiseen, muuten järjestelmän turvallisuus laskee huomattavasti [39]. Merkle-puut ovat tunnettu esimerkki siitä, miten hash-funktioita voidaan käyttää allekirjoituksissa [43]. Kuvassa 6 on esitetty esimerkki Merkle-puusta, jonka avulla voidaan allekirjoittaa neljä viestiä. $X_1 \dots X_4$ ovat avaimia, joilla salaus tehdään, ja $Y_1 \dots Y_4$ niitä vastaavia julkisia avaimia. Kulkemalla puuta alaspäin päädytään aina puun huippuun (Y_7), jonka avulla allekirjoituksen aitous voidaan varmistaa.



Kuva 6. Merkle-puu, jonka avulla voidaan allekirjoittaa 4 viestiä.

4. KRYPTOGRAFIA KÄYTÄNNÖN MAAILMASSA

Kryptografiset algoritmit määritetään yleensä matemaattisten kaavojen ja operaatioiden avulla. Algoritmien implementointi on algoritmin suunnittelusta erillinen prosessi, jossa on omat sudenkuoppansa. Toteutus voi olla virheellinen, tai siinä voi tulla esiin ongelmia, vaikka teorian tasolla algoritmi olisikin turvallinen. Algoritmien implementointi oikealla ja turvallisella tavalla on yllättävän haastavaa.

Jos omassa sovelluksessa on tarve kryptografialle, kannattaa nyrkkisääntönä pitää sitä, ettei pyörää kannata yrittää keksiä uudelleen. On olemassa lukuisia avoimen lähdekoodin kirjastoja, jotka toteuttavat tarvittavat algoritmit. Niitä käyttämällä on todennäköisesti välttänyt suuren osan implementointiin liittyvistä ongelmista. Myös yritykset käyttävät tuotteissaan avoimen lähdekoodin kirjastoja, koska se on halvempaa sekä ennen kaikkea turvallisempaa.

Jos haluaa käyttää esimerkiksi NIST:n standardointikilpailun kvanttiturvallisia algoritmeja tavallisten julkisen avaimen salausalgoritmien sijaan, on valikoima huomattavasti heikompaa. Algoritmeista on olemassa esimerkkitoteutuksia, joita kyllä voi käyttäää. Suurien ja usein käytettyjen kirjastojen tarjoama tuki kvanttiturvallisille algoritmeille on huomattavasti heikompaa. Esimerkiksi OpenSSL:n [7] perusversiossa ei ole toteutettuna ainuttakaan NIST:n standardointikilpailussa olevista algoritmeista.

4.1. Kryptografisten algoritmien toteuttaminen käytännössä

Kryptografiassa etsitään matemaattisesti todistettavissa ja tutkittavissa olevia tapoja suojata tietoa ja kommunikaatiota. Jotta algoritmia voi oikeasti hyödyntää, se täytyy ensin implementoida eli kirjoittaa jollain ohjelmointikielellä ja integroida järjestelmään, jossa sitä käytetään. Missä tahansa sovelluksessa, jossa tietoa ja kommunikaatiota suojataan, käytetään kryptografisia algoritmeja.

Implementointivaiheessa täytyy ottaa huomioon useita eri asioita. Oikeassa maailmassa pelkästään algoritmin ja toteutuksen turvallisuus ei riitä. Suorituskyvyn merkitys on suuri. Mitä enemmän operaatioita joudutaan tekemään, sitä tärkeämpää on, että algoritmi ja sen implementaatio ovat tehokkaita. Pienen suorituskyvyn omaavissa laitteissa algoritmien keveyden merkitys vain korostuu. Jos esimerkiksi haluaa IoT-laitteilla (esineiden internet, engl. *Internet of Things*) käyttää kryptografisia algoritmeja, on mahdollisten algoritmien valinta huomattavasti pienempi.

4.2. Haasteita ja vaaroja käytännön toteutuksissa

Vaikka kryptografinen algoritmi olisi matemaattisesti turvallinen, se ei tarkoita, että sen toteuttaminen tai käyttö olisi mutkatonta. Toteutus- tai käyttövaiheessa saattaa käydä kömmähdyksiä, väärinymmärryksiä tai muuten vaan virheitä, jotka vaarantavat järjestelmän turvallisuuden.

Koska myös kryptografiset ohjelmistokirjastot ovat ihmisten toteuttamia, myös niissä nousee esille vakavia haavoittuvuuksia. Yksi viime vuosien tunnetuimmista vakavista haavoittuvuuksista on Heartbleed [44], joka oli virhe OpenSSL-kirjaston TLS-toteutuksessa ja mahdollisti kriittisen informaation pääsyn hyökkääjien käsiin.

Tuoreempi esimerkki on haavoittuvuus Windowsin CryptoAPI:n elliptisen käyrän salausmenetelmien sertifikaattien vahvistamisessa [45]. Kyseinen haavoittuvuus mahdollisti esimerkiksi sen, että hyökkääjä kykeni saamaan vahingollisen tiedoston näyttämään olevan luotetusta lähteestä.

Vaikka kryptografiset algoritmit olisi implementoitu oikealla tavalla ja turvallisesti, niiden käyttö voi osoittua hankalaksi. Vuodelta 2014 peräisin olevassa tutkimuksessa tutkituista haavoittuvuuksista 17 % johtui kryptografisissa kirjastoissa tehdyistä virheistä, ja jopa 83 % kirjastoja käyttävistä sovelluksista, johtuen esimerkiksi kirjaston vääränlaisesta käytöstä [46]. Tämä osoittaa, että kryptografisten kirjastojen järkevä suunnittelu ja selkeät rajapinnat ovat äärimmäisen tärkeitä. Eräässä tutkimuksessa vuodelta 2013 huomattiin, että 88%:ssa yli 10 000 tutkitusta Google Play -kaupassa olleesta Android-sovelluksesta löytyi vähintään yksi virhe kryptografisten rajapintojen käytössä [47].

Kryptografisten algoritmien käytössä on helppo tehdä virheitä, vaikka algoritmit olisivat oikein toteutettu. Esimerkiksi satunnaisuuden kanssa täytyy järjestelmän implementoijan olla hyvin tarkkana. On tärkeää, että satunnaisuuden luomisessa käytetään kryptografisesti vahvoja satunnaislukugeneraattoreita.

4.3. Kvanttiturvallisten algoritmien toteuttaminen

Kvanttiturvallisten kryptografisten algoritmien implementointi ei käytännössä eroa tavallisten kryptografisten algoritmien implementoinnista, sillä kyseessä ovat vain erilaisiin matemaattisiin ominaisuuksiin perustuvat algoritmit. Koska kvanttiturvalliset algoritmit suoritetaan samanlaisilla tietokoneilla kuin muutkin algoritmit, suurin osa samoista haasteista ja vaikeuksista perinteisten algoritmien implementoinnissa pätevät myös kvanttiturvallisiin algoritmeihin. Suuri haaste kvanttiturvallisten algoritmien implementoinnissa on kuitenkin algoritmien monimutkaisuus, ja niiden spesifikaatioissa oleva todella korkean tason matematiikka, jolloin algoritmin implementointi tyhjästä pelkän spesifikaation pohjalta on tavalliselle ohjelmoijalle on todella vaikeaa [48]. Julkisen avaimen kvanttiturvalliset algoritmit perustuvat huomattavasti vaikeammin ymmärrettävän matematiikkaan kuin perinteiset algoritmit, kuten RSA.

Toinen suuri haaste kvanttiturvallisten algoritmien implementoinnissa on suuri avaintenkoko ja osassa algoritmeista myös suuri tuloksena syntyvän salatekstin koko. Taulukossa 3 on vertailtu tässä työssä toteutettujen kvanttiturvallisten algoritmien ja perinteisten algoritmien avaintenkokoja. Samalla turvallisuustasolla varsinkin kvanttiturvallisten algoritmien yksityiset avaimet ovat suurikokoisia. Laitteistopohjaisissa toteutuksissa näiden haasteiden merkitys vain korostuu. Muitakin haasteita nousee esille: satunnaisuuden luominen oikealla tavalla on tärkeää. Koska kvanttiturvalliset käyttävät erilaisia matemaattisia operaatioita kuin perinteiset algoritmit, laitteiston optimointi algoritmia varten hankaloituu [48].

Jotta kvanttiturvallisia algoritmeja olisi helppo liittää omiin toteutuksiin, tai korvata jokin ei-kvanttiturvallinen algoritmi jo valmiissa toteutuksessa, on tärkeää, että kvanttiturvallisten algoritmien toteutuksia on helposti saatavilla. Taulukossa 4 on eritelty julkisen avaimen kvanttiturvallisten algoritmien saatavuutta tämän tekstin kirjoittamishetkellä osassa suurimmista avoimen lähdekoodin kryptografisissa

Taulukko 3. Kvanttiturvallisten algoritmien ja perinteisten julkisen avaimen salausmenetelmien avainten ja salatekstin kokojen vertailua.

Algoritmi	Yksityinen avain (tavuja)	Julkinen avain (tavuja)	Salateksti (tavuja)
Kyber-512	1632	800	768
Kyber-768	2400	1184	1088
Kyber-1024	3168	1568	1568
LightSaber	1568	672	736
Saber	2304	992	1088
FireSaber	3040	1312	1472
RSA-3072	384 (modulo)	384 (modulo)	384
RSA-7680	960 (modulo)	960 (modulo)	960
RSA-15360	1920 (modulo)	1920 (modulo)	1920
Curve25519	251	256	

ohjelmistokirjastoissa. Isossa osassa avoimen lähdekoodin suosituista kirjastoista ei ole ainuttakaan kvanttiturvallista julkisen avaimen salausalgoritmia saatavilla. Syitä tähän on monia. NIST:n standardointiprosessi on tämän tekstin kirjoittamishetkellä vielä kesken, ja monet varmasti odottavat tuloksia siitä, mitkä algoritmit standardiksi lopulta päätyvät. Samoin esimerkiksi algoritmien parametrit ovat eläneet kilpailun aikana, ja algoritmeihin voi tulla vielä muutoksia. Tällöin implementointeja jouduttaisiin mahdollisesti tekemään useampaan otteeseen. Riskinä on myös se, että koska useat algoritmit ovat nuoria ja edelleen tarkan tutkimuksen alla, löytyy joihinkin niistä hyökkäyksiä, jotka laskevat tuntuvasti niiden turvallisuustasoja.

Taulukko 4. Kvanttiturvallisten asymmetristen algoritmien saatavuus erinäisissä avoimen lähdekoodin ohjelmistokirjastoissa työn tekemisen aikana.

Kirjasto	Kieli	Algoritmit
OpenSSL	C	-
LibreSSL	C	-
Botan	C++	McEliece, NewHope, XMSS
Crypto++	C++	-
Bouncy Castle	Java	NewHope, McEliece, Rainbow, ...

4.4. Open Quantum Safe -projekti

Open Quantum Safe [49] on avoimen lähdekoodin projekti, jonka tarkoituksena on auttaa kvanttiturvallisen kryptografian tutkimuksessa ja käyttöönotossa. Projektissa kehitetään *liboqs*-nimistä C-kielistä ohjelmistokirjastoa [50], joka sisältää useita kvanttiturvallisia algoritmeja. Kirjastoa ei kuitenkaan suositella käytettävän tuotantoympäristössä tai oikeasti sensitiivisen datan suojaamiseen, sillä kirjaston

tarkoitus on auttaa tutkimuksessa ja prototyyppien tekemisessä. *Open Quantum Safe* -projektissa myös pyritään tekemään prototyyppejä, joissa kvanttiturvallisia algoritmeja integroidaan joukkoon protokollia ja sovelluksia, kuten esimerkiksi TLS ja SSH.

5. KVANTTITURVALLISTEN ALGORITMIEN INTEGROINTI OHJELMISTOKIRJASTOON

Uusien asymmetrisisten kvanttiturvallisten algoritmien saatavuus ohjelmistokirjastoissa on vähäistä. Niistä on olemassa yleensä C-kielisiä esimerkkitoteutuksia, ja on olemassa tiettyjä kirjastoja, jotka erikoistuvat kvanttiturvallisiin tai vastaaviin algoritmeihin. Paljon helpompaa käyttäjälle olisi, että algoritmit olisivat osa kirjastoja, joita he ovat muutenkin tottuneet ohjelmissaan käyttämään. Tällöin käyttäjän ei tarvitse opetella uusia eri kirjastojen rajapinnoille ominaisia piirteitä eikä asentaa tai ladata uusia kirjastoja sen takia että voisivat kokeilla yhtä uutta algoritmia. Jotta uusia algoritmeja pääsisivät mahdollisimman monet kokeilemaan, on tärkeää että ne ovat mahdollisimman helposti saatavilla.

Tässä työssä tavoitteena oli tutustua syvemmin muutamaan NIST:n kvanttiturvallisten algoritmien kilpailun algoritmeista. Samoin tavoitteena oli tutkia, mitä kvanttiturvallisten algoritmien liittäminen ohjelmistokirjastoon niiden ylläpitäjiltä vaatii, mitä ongelmia matkalla voi tulla vastaan, ja miten algoritmien liittäminen lopulta onnistuu. Tavoitteena oli liittää useampi NIST:n kvanttiturvallisten algoritmien kilpailun viimeisen kierroksen finalisteista ohjelmistokirjastoon ja arvioida sekä prosessia että onnistumista. Kirjastoksi valikoitui Crypto++ [8], joka on C++-kielinen varsin hyvin tunnettu avoimen lähdekoodin kirjasto, mutta jossa ei vielä tämän työn aloittamishetkellä ollut toteutettuna kilpailun algoritmeja. Tässä luvussa esitellään lyhyesti tässä työssä Crypto++-kirjaston kopioon [9] integroidut kvanttiturvalliset julkisen avaimen algoritmit, käydään läpi miten työn tekeminen onnistui ja mitä haasteita tuli vastaan matkan varrella. Kopio on GitHubista kenen tahansa ladattavissa ja testattavissa.

5.1. Algoritmien valinta

Tässä työssä integrointiin Crypto++-kirjaston kopioon yhteensä kolme NIST:n standardointikilpailun kolmannen kierroksen algoritmia. KEM-algoritmeista integroitiin CRYSTALS-Kyber [32] ja SABER [34], sekä digitaalisista allekirjoitusalgoritmeista CRYSTALS-Dilithium [33]. Kaikki valitut algoritmit perustuvat hilaongelmiin, ja se oli tietoinen valinta algoritmeja valittaessa. Koska kaikkien kolmen algoritmin taustalla oleva perusidea on sama, pystyy niiden välillä tekemään järkevää vertailua, ja samalla saamaan syvempää käsitystä siitä, miten hiloihin perustuvat algoritmit toimivat.

Algoritmien valintaan vaikutti myös se, kuinka todennäköisesti ne oltaisiin asettamassa viralliseksi standardiksi. Kaikki kolme valittua algoritmia ovat kolmannen kierroksen finalisteja eli todennäköisiä standardoitavia algoritmeja. Mikäli Kyberistä tai SABERista ei kolmannen kierroksen aikana löydetä suuria haavoittuvuuksia, on erittäin todennäköistä, että toinen algoritmeista valitaan viralliseksi standardiksi [40]. Samoin NIST:n mukaan Dilithium on yksi ensimmäisistä vaihtoehdoista standardiksi digitaalisten allekirjoitusalgoritmien joukossa. Tässä työssä Dilithiumin valintaan vaikutti myös se, että se on samoilta tekijöiltä kuin Kyber ja perustuu samaan ongelmaan. Yksi kriteeri algoritmien valinnassa oli myös se, että ne olisivat suorituskyvyltään standardointikilpailun parhaimmistoa.

5.2. Integroinnin toteuttamisen periaatteet

Integroinnin tavoitteena oli saavuttaa lopputulos, jossa kuka tahansa tässä työssä tehtyä Crypto++:n kopiota käyttämällä voisi käyttää ja testata kvanttiturvallisia algoritmeja. Seuraavassa ovat periaatteet, joita pyrittiin työssä noudattamaan:

- Käytettävyys. Käyttäjä pystyy helposti ja kirjaston muiden algoritmien kanssa mahdollisimman yhdenmukaisella tavalla ottamaan omassa koodissaan käyttöön toteutettuja kvanttiturvallisia algoritmeja. Samoin parametrien vaihtaminen tulisi olla käyttäjälle mahdollisimman helppoa ja yksinkertaista, oikeastaan käyttäjän ei pitäisi tarvita huolehtia parametreistä ollenkaan.

- Koodin yhdenmukaisuus muun kirjaston kanssa. Tavoitteena oli, että mikäli kirjaston rakenne olisi tuttu, olisi myös kvanttiturvallisten algoritmien koodi helposti omaksuttavaa. Samoin mikäli algoritmeissa käytettäviä funktioita tai rakenteita on jo toteutettu Crypto++-kirjastoon aiemmin, käytetään niitä mallitoteutusten funktioiden sijaan.

- Luotettavuus. Toteutettujen algoritmien tulee tietysti toimia luotettavasti oikealla tavalla ja palauttaa oikeat tulokset.

- Mahdollisimman pieni koodin määrä. Käytännössä tämä tarkoittaa sitä, että kvanttiturvallisessa algoritmissa käytetään mahdollisimman paljon kirjaston valmiita toteutuksia muista algoritmeista.

- Suorituskyky. Algoritmien tulee olla mahdollisimman nopeita, ilman turhaa tavaraa. Käytännössä tämä tarkoittaa sitä, että mallitoteutuksia käytetään suurimmaksi osaksi sellaisenaan. Koska C-kielinen koodi on myös yleensä toimivaa C++-koodia, ei suuria muokkauksia suurimmaksi osaksi tarvitse tehdä.

5.3. Algoritmien liittäminen ohjelmistokirjastoon

Suuri osa standardointikilpailussa olevista algoritmeista on todella monimutkaisia, matemaattisesti hyvin kompleksisia ja suurikokoisia. Ne myös hyödyntävät muita kryptografisia algoritmeja sisäisesti. Näistä syistä algoritmien implementointi tyhjästä pelkkä spesifikaatio pohjana olisi tämän työn puitteissa samassa mittakaavassa mahdotonta. Siksi tässä työssä integroinnissa pohjana käytettiin standardointikilpailuun osallistujien tekemiä valmiita esimerkkitoteutuksia, jotka sitten muokattiin soveltuvaksi Crypto++-kirjastoon sekä korjattiin integroinnissa tapahtuvia ongelmatilanteita. Näin varmistettiin, että algoritmit (kunhan integraatio on onnistuneesti tehty) varmasti toimivat ilman suurempia kummallisia bugeja, ja vältettiin implementoinnissa mahdollisesti vastaan tulevia ymmärrys- tai ajatusvirheitä, jotka pahimmillaan saattaisivat murentaa pohjan koko algoritmin turvallisuudelta.

Kaikista kolmesta algoritmista on olemassa C-kieliset esimerkkitoteutukset (saatavilla esimerkiksi NIST:n kilpailun projektisivulta [51]). Näitä esimerkkitoteutuksia käytettiin tässä työssä algoritmien liittämiseksi valittuun

kirjastoon. Tavoitteena oli, että tässä työssä kehitettävää kirjaston kopiota käyttämällä käyttäjät voisivat helposti omissa toteutuksissaan käyttää kvanttiturvallisia algoritmeja tavalla, joka vastaa kirjaston muita algoritmeja. Samaten tavoitteena oli, että eri parametrien käyttäminen algoritmeissa (toisin sanoen algoritmien eri turvallisuustasot) olisi mahdollisimman helppoa ja yksinkertaista. Toteutuksessa pyrittiin myös alustariippumattomuuteen, ainakin siinä määrin, että algoritmien käyttäminen on mahdollista sekä Linuxilla että Windowsilla. Koko operaation oli tarkoitus myös toimia henkilökohtaisena oppimisprosessina sekä C++:n, Crypto++-ohjelmistokirjaston että valittujen algoritmien suhteen.

5.4. Kvanttiturvallisten algoritmien parametreistä

Hilapohjaisissa algoritmeissa on hyvin paljon erilaisia parametrejä, joita muuntelemalla voidaan vaikuttaa suuresti algoritmin turvallisuuteen ja suorituskykyyn. Myös NIST:n järjetämän kilpailun aikana eri algoritmien parametrejä on muuteltu varsin runsaasti. Osa parametreistä on muunneltavissa (niitä säätämällä algoritmin turvallisuustasoa pystyy muokkaamaan), osa taas täysin algoritmin sisäisiä eikä niitä ole tarkoitettu muutettaviksi. Havainnollistamiseksi taulukossa 5 on esitetty Kyberin kolmannen kierroksen parametrit.

Taulukko 5. Kyberin eri versioiden parametrit NIST:n kilpailun kolmannella kierroksella.

Algoritmin versio	n	k	q	η_1	η_2	(d_u, d_v)	δ
Kyber-512	256	2	3329	3	2	(10,4)	2^{-139}
Kyber-768	256	3	3329	2	2	(10,4)	2^{-164}
Kyber-1024	256	4	3329	2	2	(11,5)	2^{-174}

Taulukossa n ja q ovat parametrejä Kyberissä käytettävissä polynomirenkaissa $\mathbb{Z}[X]/(X^n + 1)$ ja $\mathbb{Z}_q[X]/(X^n + 1)$. n on asetettu arvoon 256, koska entropian määrä kapseloinnissa on 256 bittiä. q on valittu tietyllä tavalla sen mahdollistamiseksi, että NTT:tä (engl. *Number Theoretic Transform*) voidaan käyttää polynomien kertomiseen. NTT on diskreetin Fourier-muunnoksen erityistapaus. Sen käyttäminen nopeuttaa algoritmin toimintaa. k:n avulla pääasiassa muutetaan Kyberin turvallisuustasoja. k asettaa hilan dimension n:n moninkerraksi. η_1 ja η_2 määrittävät kohinaa algoritmeissa. d_u ja d_v käytetään salatekstin pyöristämiseen ja ne siten vaikuttavat salatekstin kokoon. δ on parametrien perusteella määrittyvä todennäköisyys, että oikeanlaisen kapseloidun salatekstin avaus epäonnistuu. [52]

5.5. Algoritmien turvallisuus

Algoritmien kolmannen kierroksen spesifikaatioista löytyy analyysiä algoritmien turvallisuudesta tunnettuja hyökkäyksiä vastaan. Hila-algoritmeja vastaan on olemassa algebrallisia hyökkäyksiä, jotka perustuvat hilojen ominaisuuksiin. On olemassa erilaisia hyökkäyksiä, joilla pyritään heikentämään hila-algoritmien taustalla olevien

oletusten turvallisuutta. Myös NIST:n standardointikilpailun aikana algoritmeista on löytynyt heikkouksia, jotka vähentävät niiden turvallisuutta [40].

On useita eri mahdollisuuksia, miten valitut algoritmit voisivat olla haavoittuvaisia tai heikompia kuin tällä hetkellä uskotaan. Käytettyjen hilojen rakenteesta tai algoritmien perusrakenteesta voisi löytyä heikkouksia, samoin kuten käytetyistä parametreistä. Ajoitushyökkäykset ovat eritoten implementointeja koskeva uhka. Standardointiprosessin aikana useissa eri algoritmeissa on tehty esimerkiksi parametrien muutoksia, kun on käynyt ilmi, etteivät valitut parametrit olleetkaan niin turvallisia kuin oli oletettu [40]. Tämän tekstin kirjoittamishetkellä vaikuttaa kuitenkin siltä, että sekä Kyberin, Dilithiumin että SABERin rakenne on turvallinen. Kaikkia algoritmeja kuitenkin tutkitaan aktiivisesti standardoimisprosessin aikana, ja on mahdollista, että niistä löytyy heikkouksia, jotka vaikuttavat standardointipäätöksiin.

5.5.1. Algoritmien turvallisuustasot

NIST:n standardointikilpailuun osallistujat lähettävät algoritmeistaan turvallisuudeltaan erovia versioita. Käytännössä tämä tarkoittaa sitä, että parametrit eroavat eri versioiden välillä. NIST:n standardointiprosessissa on määritelty viisi eri turvallisuustasoa [53], jotka on esitetty taulukossa 6 heikoimmasta vahvimpaan.

Taulukko 6. Algoritmien turvallisuustasot NIST:n standardointiprosessissa.

Turvallisuustaso	Kuvaus
I	Yhtä vaikea tai vaikeampi rikkoa kuin AES-128
II	Yhtä vaikea tai vaikeampi rikkoa kuin SHA-256
III	Yhtä vaikea tai vaikeampi rikkoa kuin AES-192
IV	Yhtä vaikea tai vaikeampi rikkoa kuin SHA-384
V	Yhtä vaikea tai vaikeampi rikkoa kuin AES-256

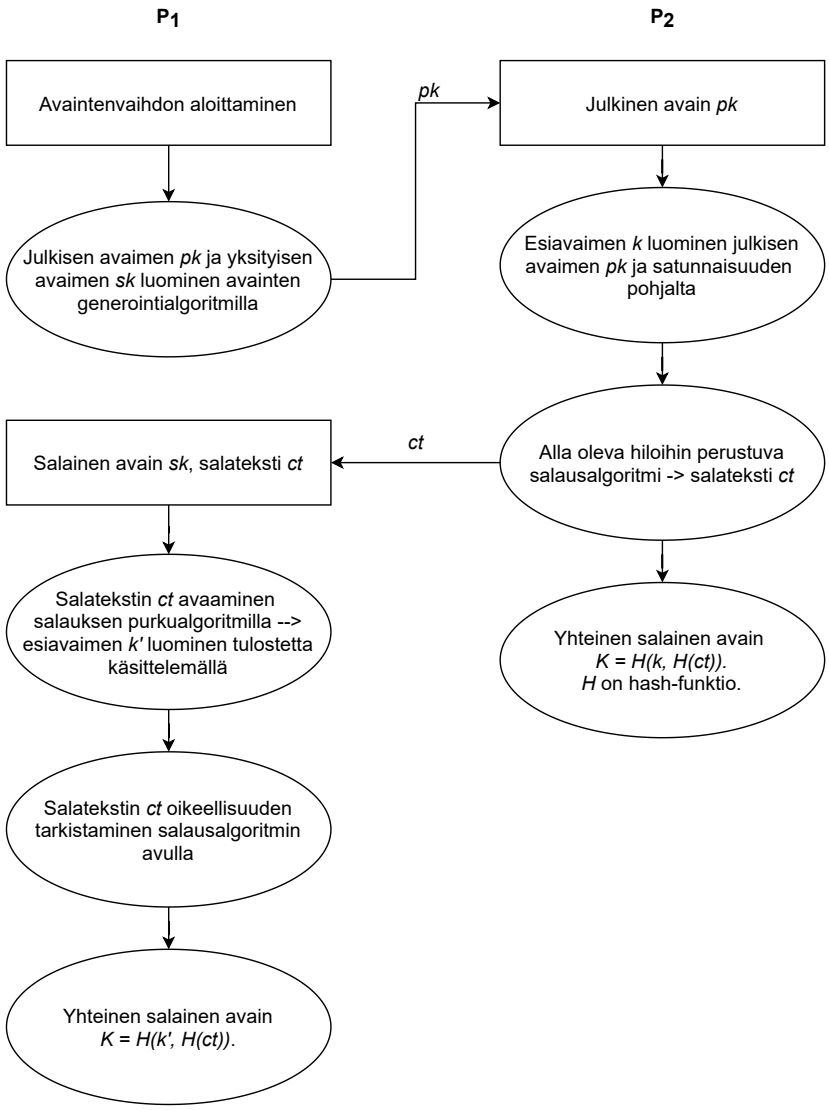
Esimerkiksi Kyberin versiot Kyber-512, Kyber-768 ja Kyber-1024 vastaavat turvallisuustasoja I, III ja V. Standardointiprosessin aikana useassa algoritmissa parametrejä ja tiettyjä sisäisiä rakenteita on jouduttu muuttamaan, kun on käynyt ilmi, ettei turvallisuus enää vastannutkaan esimerkiksi uuden heikkouden löytymisen myötä ilmoitettua turvallisuustasoa.

5.6. Ensimmäinen KEM-algoritmi - Kyber

Ensimmäinen tässä työssä toteutettu algoritmi oli CRYSTALS-Kyber [32]. Kyber perustuu *Module learning with errors* (MLWE) -ongelmaan, joka pohjautuu *learning with errors* -ongelmaan [31]. MLWE-ongelma on varsin uusi, mutta sitä vastaan ei ole löydetty hyökkäyksiä, jotka eivät pätisi myös LWE-ongelmaan [40]. Kyber on CCA-turvallinen (*Chosen Ciphertext Attack*) KEM-algoritmi. Kyber perustuu CPA-turvalliseen (*Chosen Plaintext Attack*) julkisen avaimen salausjärjestelmään, joka Fujisaki-Okamoto -muunnoksella [54] muutetaan CCA-turvalliseksi. Kyberin rakenne

on suunniteltu siten, että se mahdollistaa NTT:n käyttämisen laskuoperaatioiden nopeuttamiseksi.

Kuten kappaleessa 3.3 kuvattiin, KEM-algoritmit koostuvat kolmesta osasta, avainten generointialgoritmista, salaus- ja salauksen purkamisalgoritmeista. Kyberin avainten generointialgoritmi luo julkisen ja yksityisen avaimen käyttäjälle parametrien mukaan. Jos Kyberiä käytetään esimerkiksi juurikin avaintenvaihtoon, ensin ensimmäinen osapuoli luo julkisen ja yksityisen avaimen. Sen jälkeen toinen osapuoli voi käyttää tätä luotua julkista avainta salausalgoritmiin, joka julkisen avaimen pohjalta luo salaisen avaimen sekä salatekstin. Lopuksi ensimmäinen osapuoli voi käyttää salauksen purkamisalgoritmia, joka ottaa osapuolen oman yksityisen avaimen ja toisen osapuolen luoman salatekstin ja tulostaa saman salaisen avaimen kuin mikä on toisellakin osapuolella. Näin molemmilla osapuolilla on hallussaan sama salainen avain, jota he voivat esimerkiksi käyttää symmetrisessä salausjärjetelmässä. Kuvassa 7 on esitetty avaintenvaihto yksinkertaistettuna.



Kuva 7. Yksinkertaistettu kuvaus avaintenvaihdosta osapuolien P_1 ja P_2 välillä Kyberiä käyttäen.

5.6.1. Kyberin toteuttaminen tässä työssä

Kyber on toteutuksena todella isokokoinen ja monimutkainen sekä hyödyntää useita muita eri algoritmeja. Kyberistä on saatavilla algoritmin suunnittelijoiden tekemä C-kielinen esimerkkitoteutus, jota käytettiin tässä työssä perustana. Tämä esimerkkitoteutus siirrettiin Crypto++-kirjastosta tehtyyn kopioon ja muokattiin C++-koodiksi. Käytännössä yksinkertaisimmallaan tämä tarkoitti esimerkiksi sopivan luokkarakenteen lisäämistä koodiin ja metodien ja muuttujien nimien johdonmukaista muokkaamista luettavammiksi.

Yksi suurimmista haasteista oli sen selvittäminen, miten vaihtuvien parametrien käsittely olisi järkevintä tehdä. Esimerkkitoteutuksessa Kyberin parametrit on toteutettu kääntäjävakioiden avulla, ja koska osa niistä vaihtelee eri Kyberin versioiden välillä, ei niitä suoraan sellaisenaan voinut käyttää integroidussa versiossa. Kyberin kohdalla tässä työssä ratkaisuna käytettiin luokkamalleja (engl. *template*) [55], joiden avulla luotiin Kyberin perusluokka, joka vaatii kaksi malliparametria voidakseen määrittää muut vakiot niiden avulla. Kun käyttäjä haluaa käyttää jotain Kyberin versiota, kutsuu hän perusluokan aliluokkaa, joka määrittää luokkamallin parametrien arvot vastaamaan sen turvallisuustasoa.

Muuten suurimmat muutokset koodiin luokkarakenteen lisäämisen ohella olivat kohdat, missä Kyberin käyttäessä jotain muuta algoritmia vaihdettiin integroitu versio käyttämään Crypto++-kirjaston omia valmiita implementointeja. Koska Kyber oli ensimmäinen algoritmi, joka tässä työssä liitettiin kirjastoon, kului sen toteuttamisessa ajallisesti pisin aika. Lopputuloksen Kyberin kaikki kolme eri versiota ovat käyttäjän helposti käytettävissä, ja kaikki toimivat luotettavasti palauttaen oikean tuloksen. Yksinkertaiset testit toimivat ilman ongelmia.

5.7. Digitaalinen allekirjoitusalgoritmi - Dilithium

Dilitihium on digitaalinen allekirjoitusalgoritmi, ja kuuluu samaan algoritmiperheeseen kuin Kyber. Dilithiumia on yksityiskohtaisemmin esitelty esimerkiksi NIST:n esittelymateriaaleissa [33]. Tässä kappaleessa esitellään Dilithiumin perusteita kevyemmällä otteella. Dilithium koostuu kolmesta osasta:

- Avainten generointi. Avainten generointialgoritmi luo $k \times l$-kokoisen matriisin A. k ja l ovat Dilithiumin parametrejä, jotka ovat eri suuruisia eri turvallisuustasoilla. Samaten generointialgoritmi luo avainvektorit s_1 ja s_2, joita käytetään julkisen avaimen toisen osan luomiseen.

- Allekirjoittaminen. Allekirjoitusalgoritmi käyttää polynomilaskentaa ja avainten luomisvaiheessa luotuja matriisia A ja avainvektoreita s_1 ja s_2 allekirjoituksen tekemiseen. Allekirjoituksen pitää täyttää tietyt ehdot ollakseen oikeanlainen ja ettei se vuoda tietoa ulkopuolelle, minkä seurauksena operaatioita usein joudutaan tekemään useampi kierros. Tämä on algoritmin suunnittelussa otettu huomioon, ja parametrit on suunniteltu siten, ettei kierroksia todennäköisesti kovin montaa tule.

- Allekirjoituksen todentaminen. Algoritmi varmistaa, että allekirjoitus täsmää viestin sisältöön. Käytännössä algoritmi varmistaa, että allekirjoitus täyttää tietyt matemaattiset ehdot, joiden perusteella voidaan luottaa siihen, että allekirjoitus on oikeanlainen.

Dilithiumin operaatiot tapahtuvat polynomirenkaassa $\mathbb{Z}_q[X]/(X^n + 1)$, missä $q = 2^{23} - 2^{13} + 1$ ja $n = 256$. Dilithium on mahdollista toteuttaa deterministisenä tai probabilistisenä algoritmina. Ainoa ero näiden kahden vaihtoehdon välillä on se, että hankitaanko satunnaisuus viestin ja avaimen kautta vaiko täysin satunnaisesti. Dilithiumissa myös hyödynnetään NTT:tä laskuoperaatioiden nopeuttamiseksi.

Dilithium on suorituskyvyltään tasaisen vahva kaikilta osilta. Myöskään avainten ja allekirjoitusten koko ei ole suurimmasta päästä. NIST:n standardointiprosessissa Dilithiumin suurin kilpailija on FALCON, joka on myös hilapohjainen allekirjoitusalgoritmi. Dilithiumin rakenne mahdollistaa yksinkertaisemman rakenteen kuin FALCONin kohdalla, toisaalta taas FALCONin allekirjoituksen ja julkisen avaimen koko on pienempi. [40]

5.7.1. Dilithiumin toteuttaminen tässä työssä

Dilithiumista oli samaten saatavilla C-kielinen mallitoteutus, jota käytettiin pohjana. Aivan kuten samaan algoritmiperheeseen kuuluva Kyber, Dilithiumin mallitoteutus hyödyntää kääntäjävakioita parametrien tallentamiseksi. Koska useat Dilithiumin parametreistä vaihtelevat eri Dilithiumin versioiden välillä, esimerkkitoteutuksessa eri Dilithiumin versioiden välillä vaihtaminen lennosta ei ole mahdollista, ellei sitten ole sattunut kääntämään erikseen kaikkia neljää eri Dilithiumin versiota. Siksi suurin yksittäinen haaste Dilithiumin siirtämisessä C++-koodiksi oli, kuinka järkevästi esittää ja toteuttaa parametrit.

Yksi vaihtoehto olisi ollut toteuttaa tämä luokkamallien avulla, aivan kuten Kyberin kohdalla. Luokkamallit eivät kuitenkaan ole aivan ongelmattomia. Luokkamallia käyttävien luokkien funktiot täytyy olla määritelty otsikkotiedostossa tai sitten eksplisiittisesti täytyy C++-tiedostossa olla määritelty, mitä mahdollisia arvoja luokkamalli voi saada. Koska Dilithiumissa vaihtuvia parametreja on todella paljon, koodin siistiyden takia tämä vaihtoehto jätettiin huomiotta.

Lopulta Dilithiumin parametrit toteutettiin Dilithium-luokan sisäisinä jäsenmuuttujina, joista eri Dilithiumin mukaan versioiden vaihtuvat määrittää Dilithiumin aliluokka Dilithium-luokan rakentajafunktiota kutsuttaessa. Tämä ratkaisu aiheutti tiettyjä muita muutoksia. Mallitoteutuksessa polynomit ja vektorit on toteutettu struct-rakenteilla, joissa taulukkojen koot on määritelty edellä mainituilla parametreillä. Koska tässä työssä toteutetussa versiossa Dilithiumista kääntäjä ei kääntäessä voi tietää kaikkien parametrien kokoa, polynomien ja polynomivektorien toteutus täytyi muuttaa tässä työssä. Polynomit vaihdettiin toimimaan C++-standardikirjaston *std::array*n avulla, ja vektorit *std::vector*in avulla. Vaikka tämä aiheutti tietenkin joitakin muutoksia funktioihin, muutos oli loppujen lopuksi varsin yksinkertainen, sillä *std::array* ja *std::vector* mahdollistavat varsin helposti niiden sisäisen datan käsittelyn.

Lopputuloksena Dilithiumista oli toteutettuna neljä erivahvuista versiota, ja testien perusteella ne toimivat ongelmattomasti. Toteutuksen evaluointivaiheessa kuitenkin kävi ilmi, että Dilithiumia oli jonkin verran muutettu mallitoteutuksen lataamisen jälkeen. Dilithiumin päivittäminen jäi kuitenkin myöhemmäksi ajankohdaksi.

5.8. Toinen KEM-algoritmi - SABER

SABER [34] on Kyberin ja Dilithiumin tavoin hilapohjainen algoritmi. Aivan kuten Kyber, SABER on KEM-algoritmi, eli sitä on tarkoitus käyttää avaintenvaihdossa. SABER myös koostuu samanlaisista perusosista kuin Kyberkin. SABER perustuu *Module learning with rounding* -ongelmaan (MLWR), joka on läheistä sukua LWE-ongelmalle. MLWR-ongelman käyttö algoritmin pohjana pienentää avainten kokoa, sillä LWE-ongelmaan perustuvat algoritmit tarvitsevat satunnaisuutta, joka pitää liittää myös osaksi avaimia. MLWR-ongelma eroaa siinä mielessä MLWE-ongelmasta, että virheen lisääminen algoritmissa on korvattu pyöristämisellä [40].

SABERin suunnittelussa on tehty muitakin valintoja, joiden seurauksena se on kevyempi ja yksinkertaisempi kuin esimerkiksi Kyber tai Dilithium. Kaikki modulot ovat kahden potensseja, minkä seurauksena ohjelmisto- sekä laitteistopohjaisten toteutusten tekeminen on yksinkertaisempaa ja vakioajassa tapahtuvien implementointien suunnittelu on mahdollista.

Avaintenvaihto SABERin avulla tapahtuu korkealla tasolla tarkasteltuna lähes täysin samalla tavalla kuin Kyberinkin avulla. SABER ja Kyber ovat molemmat hyvin läheistä sukua oleviin ongelmiin perustuvia KEM-algoritmeja, joten niiden toiminta korkealla tasolla eroaa hyvin vähän. Suurimmat erot algoritmien välillä tulevatkin esille korkean tason KEM-toiminnallisuuden alla olevissa salaus- ja salauksen avausalgoritmeissa. SABERissa ne ovat huomattavasti yksinkertaisemmat suunnitteluvaiheessa tehtyjen valintojen myötä. Kyberin etu taas on se, että siinä, samoin kuten Dilithiumissa, voidaan käyttää polynomilaskennan nopeuttamiseksi NTT-muunnosta. SABERin suorituskyky on kuitenkin erinomainen kilpailun algoritmien joukossa, sillä sen rakenne mahdollistaa muilla tavoin operaatioiden tehokkaan optimoinnin [34].

5.8.1. SABERin toteutus tässä työssä

SABER oli kolmas ja viimeinen tässä työssä liitetty algoritmi. Osittain tästä syystä työskentelyprosessi oli kaikista helpoin ja yksinkertaisin. Aiempien algoritmien liittämisessä vastaan tulleiden ongelmien ratkaisuja oli helppo hyödyntää myös SABERin kohdalla.

SABERin toteutuksen pohjana käytettiin jälleen kerran C-kieleistä mallitoteutusta. SABERin mallitoteutus on huomattavasti pienempi ja yksinkertaisempi kuin Kyberin tai Dilithiumin vastaava. Siten myös Saberin integrointi oli nopeampaa. SABERin integroinnissa käytettiin hyödyksi luokkamalleja parametrien määrittämisessä kuten Kyberinkin kohdalla. Muuten SABERin integrointi kirjaston yhteyteen oli kaikista yksinkertaisinta, ja koodimuutosten tarve oli vähäisin. Tämä saattaa johtua myös

siitä, että SABER oli viimeinen algoritmi joka tässä työssä integroitiin kirjastoon, ja oppimisprosessin seurauksena integrointiprosessi oli helpompi.

Lopputuloksena SABERista toteutettiin kaikki kolme versiota - *LightSaber*, *Saber* ja *FireSaber* - jotka vastaavat turvallisuustasoja 1, 3 ja 5 NIST:n asteikolla. Käyttäjän kannalta SABERin ja Kyberin käyttäminen on identtistä, metodit ovat nimetty identtisesti ja toimivat korkealla tasolla täysin samalla tavalla - ainoastaan luokka muuttuu.

5.9. Ongelmia ja haasteita

Varsinkin Kyber ja Dilithium ovat perinteisiin algoritmeihin verrattuna monimutkaisia ja kompleksisia algoritmeja, joten niiden toteuttaminen täysin tyhjästä on varmasti varsin iso haaste kokeneellekin ohjelmoijalle ja vaatii varsin syvällistä matemaattista ymmärrystä. Myös mallitoteutukset ovat varsin monimutkaisia, eikä niitä voinut aivan suoraan vain liimata ohjelmistokirjaston kylkeen ilman ongelmia. SABER on huomattavasti yksinkertaisempi rakenteeltaan, ja siten myös sen toteuttaminen on helpompaa sekä sen koko selvästi pienempi. Kuitenkin on reiluuden vuoksi todettava, että esimerkiksi NIST:n standardointikilpailussa digitaalisista allekirjoitusalgoritmeista FALCON on monimutkaisempi algoritmi toteuttaa kuin Dilithium [40]. Kaiken kaikkiaan iso osa hilaongelmiin perustuvista algoritmeista on varsin monimutkaisia.

Vaihtelevat parametrien koot saattavat hankaloittaa eri versioiden implementointia huomattavasti. Koska mallitoteutuksia ei ole suunniteltu siten, että parametrejä voisi helposti muokata tai vaihtaa toisesta ohjelmasta kutsumalla, sopivan ratkaisun löytäminen oli aluksi haasteellista. Tätä työtä tehdessä tapahtuikin paljon oppimista C++-kielen suhteen, mikä oli yksi henkilökohtaisen tason tavoitteista.

Kryptografiset ohjelmistokirjastot ovat isokokoisia. Uusien algoritmien lisäämisessä kirjastoon voi olla aika suurikin vaiva, kun kaikki koodin kääntämiseen liittyvät tiedostot täytyy päivittää juuri oikealla tavalla. Esimerkiksi Crypto++-kirjaston kohdalla ei tarkkoja ohjeita kontribuoimiseen helposti löytynyt, vaan joutui itse kokeilemalla selvittämään, kuinka uuden algoritmin lisääminen onnistuu oikealla tavalla.

Algoritmien ollessa suuria kokonaisuuksia, joissa dataa käsitellään erilaisilla matemaattisilla operaatioilla, oli debuggaus välillä hankalaa. Integrointivaiheessa ilmentyneitä bugeja sai välillä metsästää varsin pitkään suurennuslasin kanssa. Koska esimerkiksi avainten luominen vaatii satunnaisuutta, oli välillä suuria haasteita selvittää, miksi ohjelman ajamalla saatu tulos ei ollutkaan oikeanlainen. Debuggausta varten käytettiin determinististä "satunnaislukugeneraattoria", jolla saatiin aina sama tuloste. Näin mallitoteutuksen ja integroidun version tulosteita vertaamalla pystyttiin selvittämään, onko integraatiovaiheessa tullut virheitä. Käytännössä debuggausprosessi tapahtui siten, että sekä mallitoteutusta että integroitua C++-toteutusta ajettiin tiettyyn pisteeseen ja verrattiin niiden tilaa keskenään. Tällä tavalla eroavuudet toiminnassa ja lopulta virhepaikat löytyivät.

GCC-kääntäjä [56] sallii taulukoiden koon määrittämisen muuttujia käyttäen, mikä ei ole C++-standardin mukaista. C++-standardin mukaan taulukoiden määrittäminen tapahtuu vakioiden (engl. *constant*) kautta. Tämän takia koodi, joka kääntyy GCC:n

avulla, ei välttämättä käänny muilla kääntäjillä ollenkaan. Esimerkiksi Visual Studio -ympäristön kääntäjä ei tue muuttujien käyttämistä taulukoiden koon määrittämiseen. Tämän takia monessa kohtaa koodia taulukko täytyi vaihtaa *std::vector*in käyttöön, koska tavoitteena oli, että kirjasto kääntyisi mahdollisimman monella alustalla ja eri kääntäjillä.

6. TOTEUTUKSEN ANALYSOINTI

Tässä luvussa keskitytään tehdyn työn arviointiin ja Crypto++-kirjaston kopioon integroitujen algoritmien suorituskyvyn vertailuun sekä toistensa kanssa että verrattuna pohjana toimineisiin mallitoteutuksiin. Tässä luvussa lyhykäisesti myös pohditaan tässä työssä tehdyn toteutuksen turvallisuutta.

6.1. Toteutuksen suorituskyvyn analysointi

Algoritmien suorituskykyä analysoitiin mittaamalla prosessorin kellosyklejä *__rdtsc()*-käskyn avulla. Kaikki mittaukset tehtiin kannettavalla tietokoneella, jonka käyttöjärjestelmä oli Ubuntu ja ohjelmat käännettiin GCC-kääntäjällä käyttäen sekä Crypto++-kirjaston että mallitoteutusten valmiita oletuskääntämisasetuksia. Erilaisessa ympäristössä ja kääntäjän optimoinnilla tuloksissa voi olla eroja. Eri kääntäjiä käyttämällä tuloksissa voisi myös olla eroa.

Taulukossa 7 on esitelty käsiteltyjen KEM-algoritmien, Kyberin ja SABERin, mitattuja suoritusaikoja, sekä verrattu tässä työssä tehdyn toteutuksen suorituskykyä alkuperäisiin mallitoteutuksiin. Taulukosta nähdään, että Kyberin versio integroidussa toteutuksessa on kaikilla parametreillä hieman SABERin vastaavaa versiota nopeampi. Erityisesti NIST:n turvallisuustason 5 algoritmeissa (Kyber-1024 ja vastaavasti *FireSaber*) on nähtävissä selkeää eroa suorituskyvyn suhteen. Mallitoteutuksissa taas SABER peittoaa Kyberin suoritusajoissa. Säännönmukaisesti integroidut versiot algoritmeista ovat hitaampia kuin mallitoteutukset. Integroidussa versiossa SABERin kohdalla suoritusajat jopa tuplaantuvat. Kyberin kohdalla suoritusajan kasvu integroidussa versiossa on pieni verrattuna Saberiin. Kyberin kohdalla suoritusajan kasvu oli keskimäärin n. 18 % toimintoa kohden. Erot suoritusajassa Kyberin kohdalla kasvoivat algoritmien turvallisuustason ja samalla suoritusajan kasvaessa. Suurimmat erot integroidun toteutuksen ja mallitoteutuksen välillä olivat Kyber-1024:n kohdalla.

Taulukossa 8 on eritelty Dilithiumin suoritusaikoja eri parametreilla, ja verrattu suoritusaikoja Dilithiumin mallitoteutukseen. Työn tekemisen aikana Dilithiumiin ehdittiin tehdä muutoksia, ja tässä työssä toteutettu Dilithiumin versio ei aivan täysin vastaa uusinta versiota. Sen suorituskykyä verrattiin Dilithiumiin vastaavaan, jo vanhentuneeseen versioon. Taulukosta 8 on selvästi nähtävissä, että integroitu versio on hitaampi, mutta suoritusajan lisääntyminen on suurimmillaan n. 20 % operaatioissa. Suhteellinen ero suoritusajassa Dilithiumin kohdalla on pienempi suurilla parametreillä suoritusajan kasvaessa.

6.1.1. Mahdollisia vaikutuksia suorituskykyyn

Algoritmien integrointi Crypto++-kirjaston kopioon mallitoteutusten perusteella säännönmukaisesti kasvatti suoritusaikoja. Vaikka C:llä ja C++:lla kirjoitettujen ohjelmien suorituskyvyssä ei ole suuria eroja, integrointiprosessissa on tehty ratkaisuja, jotka todennäköisesti osaltaan vaikuttavat negatiivisesti ohjelman suorituskykyyn. C++-kielessä on rakenteita, jotka ovat hitaampia kuin C:n vastaavat rakenteet, ja integrointia tehtäessä päämääränä ei ollut mahdollisimman optimoitu

Taulukko 7. Tässä työssä toteutetettujen Kyberin ja Saberin integrointien suoritusajat eri parametreillä. Suoritusaikoja on verrattu C-kielisen mallitoteutuksen samalla tietokoneella mitattuihin suoritusaikoihin.

Algoritmi	Toiminto	CPU-syklit (integrointi)	CPU-syklit (mallitoteutus)	Suoritusajan kasvu (%)
Kyber-512	Avainparin luominen	77452	70527	10
	Salaaminen	103454	90177	15
	Salauksen avaaminen	124722	106539	17
Kyber-768	Avainparin luominen	137032	119089	15
	Salaaminen	170474	141759	20
	Salauksen avaaminen	196737	162677	21
Kyber-1024	Avainparin luominen	213370	181525	18
	Salaaminen	251775	205682	22
	Salauksen avaaminen	283941	230457	23
LightSaber	Avainparin luominen	84330	43321	95
	Salaaminen	106420	56580	88
	Salauksen avaaminen	121883	62496	95
Saber	Avainparin luominen	158330	80317	97
	Salaaminen	193770	100271	93
	Salauksen avaaminen	218243	109088	100
FireSaber	Avainparin luominen	257348	132859	94
	Salaaminen	305424	156832	95
	Salauksen avaaminen	340069	169193	101

lopputulos vaan mahdollisimman helppo luettavuus ja käytettävyys. Käytettävyyden kannalta algoritmien jakaminen luokkiin, joita käyttäjä eri parametreillä voi kutsua, on ehdottomasti positiivinen asia. Mallitoteutukset eivät kuitenkaan kaikki suoraan soveltuneet luokkarakenteeseen ilman muutoksia. Integroinnissa on myös pyritty mahdollisimman paljon käyttämään Crypto++-kirjaston valmiita toteutuksia, jolloin suorituskyky saattaa hävitä mallitoteutuksille.

Jos toteutuksia haluaisi tarkemmin optimoida, pitäisi paneutua yksittäisten metodien suoritusaikoihin ja se ei ole tämän työn raameissa mahdollista. Tässä työssä on toteutettu yleisversiot algoritmeista, joiden olisi tarkoitus toimia luotettavasti millä tahansa alustalla. Algoritmeista on olemassa myös mallitoteutuksia, jotka on optimoitu tietynlaiselle laitteistolle ja käyttävät laitteistokohtaisia käskyjä vielä paremman suorituskyvyn saavuttamiseksi. Ympäristöissä, joissa suorituskyky on kaikki kaikessa, on laitteistokohtainen optimointi suuressa osassa kvanttiturvallisten algoritmien osalta.

Taulukko 8. Tässä työssä toteutetun Dilithiumin integroinnin suoritusaikoja eri parametreillä. Suoritusaikoja on verrattu C-kielisen mallitoteutuksen samalla tietokoneella mitattuihin suoritusaikoihin.

Algoritmi	Toiminto	CPU-syklit (integrointi)	CPU-syklit (mallitoteutus)	Suoritusajan kasvu (%)
Dilithium1	Avainparin luominen	110487	91859	20
	Allekirj. tekeminen	525996	442746	19
	Allekirj. avaaminen	121476	100547	21
Dilithium2	Avainparin luominen	182150	151842	20
	Allekirj. tekeminen	929855	776018	20
	Allekirj. avaaminen	189708	157029	21
Dilithium3	Avainparin luominen	265158	237244	12
	Allekirj. tekeminen	1367435	1222494	12
	Allekirj. avaaminen	266249	235769	13
Dilithium4	Avainparin luominen	350147	312286	12
	Allekirj. tekeminen	1269253	1137582	12
	Allekirj. avaaminen	364606	324617	12

6.2. Toteutuksen turvallisuus

Toteutuksia analysoitiin Valgrindin [57] avulla mahdollisten muistivuotojen löytämiseksi, ja sellaisia ei löytynyt. Yksi keskeinen riski algoritmien toteuttamisessa ovat mahdolliset sivukanavahaavoittuvuudet. Niitä saattaa syntyä sekä algoritemissa itsessään että toteutuksissa olevien heikkouksien takia. Tämän työn puitteissa ei sivukanava-analyysiin ole mahdollisuutta, mutta se on aina asia, joka on pidettävä mielessä.

Ylipäätänsä algoritmia implementoitaessa on oltava tarkkana, että algoritmiin ei synny kohtia, jossa suoritusajan perusteella voitaisiin päätellä jotain informaatiota. Tämän työn toteutuksessa kiinnitettiin huomiota siihen, että käytettiin vakioajassa suoriutuvia funktioita esimerkiksi syötteen oikeellisuuden tarkistamiseen. Kyberin ja Dilithiumin rakenne aiheuttaa luonnostaan sitä, että tietyt osat algoritmista saatetaan käydä läpi useammin kuin kerran. Tämä kuitenkin on algoritmin suunnittelussa otettu huomioon, eikä sitä voi toteutusvaiheessa välttää.

7. POHDINTA

Kvanttitietokoneiden kehittyminen uhkaa erityisesti julkisen avaimen salausmenetelmien turvallisuutta. Hila-algoritmit, joihin tässä työssä tekeminen kohdistui, ovat yksi mahdollinen korvaaja esimerkiksi RSA:n ja elliptisten käyrien salausmenetelmien turvallisuuden murentuessa. Koska hila-algoritmit ovat varsin nuori joukko algoritmeja, on hyvin mahdollista, että niitä vastaan löytyy uusia hyökkäyksiä. Myös NIST:n kilpailun aikana tietyn rakenteen omaavia hila-algoritmeja vastaan on löytynyt hyökkäyksiä, mutta ne eivät todennäköisesti uhkaa tässä työssä toteutettuja algoritmeja [40].

Koska standardointiprosessi on käynnissä ja on täysin mahdollista, että eri kilpailun algoritmeja vastaan löytyy hyökkäyksiä, jotka heikentevät niiden turvallisuutta, on täysin ymmärrettävää, että niitä ei ole toteutettu kovin moneen yleisesti käytetyistä ohjelmistokirjastoista. Esimerkiksi tässä työssä toteutettu Dilithiumin versio on jo vanhentunut, sillä siihen on tehty isohkoja muutoksia kolmannen kierroksen alkamisen jälkeen. Mikäli tässä vaiheessa algoritmeja toteutettaisiin ohjelmistokirjastoihin, saattaisivat kirjaston ylläpitäjät joutua toteuttamaan algoritmin useampaan kertaan. Yleiskäyttöiset kirjastot eivät myöskään halua toteuttaa algoritmeja, joiden turvallisuudesta ei ole täysiä takeita. Todennäköistä on myös se, että ne algoritmit, jotka lopulta standardoidaan, nousevat käytetyimmiksi, ja muut kilpailun algoritmit, vaikka ne olisivat lähes yhtä hyviä ja jopa parempia tietyissä konteksteissa, jäävät unholaan. Esimerkiksi AES:n standardointikilpailuissa muut Rijndaelin kanssa kilpailleet algoritmiehdokkaat ovat jääneet hyvin pienelle käytölle, kun taas Rijndaelia käytetään nykyään kaikkialla. Siksi suurin osa varmasti odottaa NIST:n valintoja standardoitaviksi algoritmeiksi ja tarjoaa sitten toteutukset näistä.

7.1. Toteutuksen onnistuminen

Tässä työssä Crypto++-ohjelmistokirjaston kopioon liitettyjä algoritmeja voi teoriassa kuka tahansa käyttää joko harrastusmielessä tai tutkiakseen algoritmeja. Algoritmien käyttö on ainakin allekirjoittaneen mielestä yksinkertaista ja eri algoritmeja käytetään samalla tavalla, joten käytettävyystavoite täyttyy ainakin perustasolla. Algoritmitestit toimivat luotettavasti ja tuottavat oikeat tulokset, joten ainakin siihen voi luottaa, että algoritmit toimivat oikein. Myöskään testatessa algoritmitoteutuksia staattisilla ja dynaamisilla analyysityökaluilla esimerkiksi muistivuotojen varalta ongelmia ei löytynyt.

Tämä ei tietenkään tarkoita, että toteutukset olisivat täydellisiä. Algoritmien suorituskyky voisi varmasti olla parempi. Vaikka työn toteutuksessa kiinnitettiin huomiota siihen, että funktiot mahdollisuuksien mukaan suoriutuvat vakioajassa, voi toteutuksista löytyä paikkoja, jotka ovat heikkoja sivukanavahyökkäyksille. Koska vain yksi henkilö toteutti tämän työn, ei työn täydellisyyteen kannata sokeasti luottaa. Kun ohjelmistokirjastoihin oikeasti toteutetaan algoritmeja, useampi henkilö - ainakin toivottavasti - käy läpi toteutukset ja tutkii ne ongelmien varalta.

Koska toteutus perustuu C-kielellä tehtyihin esimerkkitoteutuksiin, jotka suoraan tai hieman muokkaamalla siirrettiin C++-kirjastoon, ei koodissa juuri ylimääräistä ole. Samoin aina kun algoritmeissa käytetään esimerkiksi hash-algoritmeja (mikä

tapahtuu varsin usein), on toteutuksissa pyritty käyttämään Crypto++-kirjaston sisäisiä toteutuksia niistä.

Toteutuksen suorituskykyä on arvioitu luvussa 6. Yleisellä tasolla suorituskyvyn voi ajatella olevan hyvällä tasolla, hieman hitaampi suorituskyky kuin esimerkkitoteutuksissa oli odotettavissakin. Suorituskyvyn tarkempi optimointi ei kuitenkaan ollut tämän työn puitteissa realistinen tavoite. Oikeaan käyttöön menevissä algoritmitoteutuksissa erinomainen suorituskyky on luonnollisesti yksi keskeisimmistä tavoitteista.

7.2. Haasteet kvanttiturvallisten algoritmien toteuttamisessa

Kvanttiturvallisiin algoritmeihin ja niiden toteuttamiseen liittyy useita erilaisia haasteita. Koska kvanttiturvalliset algoritmit ovat yleensä matemaattisesti huomattavasti monimutkaisempia kuin perinteiset tekijöihinjakoon tai diskreetteihin logaritmeihin perustuvat algoritmit, on niiden ymmärtäminen huomattavasti vaikeampaa. Tämän takia algoritmien implementointi on hankalampaa. Koska implementoija ei välttämättä täysin ymmärrä matemaattisia konsepteja tai algoritmin toimintaa, hyvin optimoidun ja luotettavasti toimivan implementaation toteuttaminen on suuri haaste. Algoritmien toteuttaminen ilman esimerkkitoteutuksia pelkän spesifikaation pohjalta olisi ollut tässä työssä todella hankalaa. Lopputuloksena olisi todennäköisesti ollut huomattavasti huonommin toimiva versio algoritmista, kuin mihin tässä työssä päästiin.

Esimerkiksi tässä työssä toteutetuista hilapohjaisista algoritmeista Kyber ja Dilithium ovat varsin monimutkaisia, jolloin myös implementaatiosta tulee väkisinkin monimutkainen ja -osainen. Tällöin riski siitä, että implementaatiossa tapahtuu virheitä tai vahinkoja, jotka voivat myöhemmin osoittautua turvallisuusuhkiksi, kasvaa algoritmin monimutkaisuuden mukana. SABERin suunnittelussa tähän aspektiin onkin kiinnitetty erityistä huomiota, ja se on merkittävä etu.

Tämän työn tekemisessä suurena haasteena oli toteutusten debuggaus. Algoritmeja integroidessa oli virhekohtia välillä erittäin hankalaa saada esille toteutusten monimutkaisuudesta johtuen. Koska monet hila-algoritmit eivät myöskään ole deterministisiä algoritmeja, on niiden toiminnan oikeellisuuden tarkistaminen hankalaa. Tietty syöte johtaa eri lopputuloksiin eri ajokerroilla ja niinpä voi olla hankalaa paikallistaa kohta ohjelmassa, jossa virhe tapahtuu. Algoritmien kompleksisuus vaikeuttaa debuggausta myös siten, että ei ole helppoa sanoa, mikä muuttujien tilan pitäisi olla tietyssä vaiheessa algoritmin suoritusta. Debuggauksen haasteellisuus liittyy myös vahvasti implementoinnin haastellisuuteen ylipäätänsä. Ei ole helppoa olla sataprosenttisen varma siitä, että implementoinnissa ei ole tehty virheitä.

Tässä työssä algoritmien parametrien koolla tai ohjelman tulosteen koolla ei ollut väliä. Näin ei kuitenkaan ole kaikissa ympäristöissä. Laitteissa, joissa on rajallinen muisti tai prosessointikyky, ei ole mahdollisuutta käyttää useita kvanttiturvallisia algoritmeja suurten parametrien koon takia. Myöskään kaikki kommunikaatioprotokollat eivät heti taivu kvanttiturvallisten algoritmien käyttöön esimerkiksi viestin pituuden rajoitusten takia.

7.3. KEM-algoritmien vertailua

Tässä työssä työskenneltiin kahden KEM-algoritmin, Kyberin ja SABERin, parissa. Molemmat ovat finalisteja NIST:n standardointiprosessissa ja on todennäköistä, että toinen niistä valitaan myös viralliseksi NIST:n standardiksi. Molemmissa algoritmeissa on etunsa, mutta kumpi onkaan parempi?

Avainten ja salatekstin koot Kyberissä ja SABERissa ovat hyvin lähellä toisiaan, joten niiden perusteella ei juuri valintaa kannata tehdä. Myös suorituskyvyltään algoritmit ovat hyvin samankaltaisia. Esimerkkitoteutuksista tässä työssä käytetyllä tietokoneella SABER oli nopeampi. Kuitenkin sitten tässä työssä tehdyssä implementaatiossa Kyber osoittautui nopeammaksi kuin SABER. Olisikin hyvin mielenkiintoista sukeltaa syvemmälle kysymykseen siitä, miksi näin kävi.

Kyber on rakenteeltaan monimutkaisempi kuin SABER: sen näkee jo koodin määrästä, mikä kyseisten algoritmien toteuttamiseen menee. SABERin toteutuksessa onkin keskitytty toteutuksen ja rakenteen yksinkertaisuuteen ja selkeyteen. Kyberin rakenteen etuna taas on se, että NTT:tä voidaan käyttää laskutoimitusten tehostamiseksi. SABER ei pysty käyttämään hyödyksi NTT:tä, mutta se on silti erittäin tehokas. SABERin toteutus on todennäköisesti helpompi saada suoriutumaan vakioajassa, mikä voi olla etu sivukanavahyökkäyksiä ajatellen.

Koska kummastakaan ei ainakaan tällä hetkellä ole löytynyt mittavia heikkouksia, onkin NIST haastavan kysymyksen edessä. Molemmat algoritmit ainakin tämänhetkisten tietojen perusteella sopisivat standardiksi. Ratkaisevaksi voi nousta kysymys ongelmasta, johon algoritmit perustuvat - koetaanko jompi kumpi ongelmista, MLWE tai MLWR, turvallisemmaksi valinnaksi tulevaisuutta ajatellen.

7.4. Tässä työssä käsiteltyjen algoritmien soveltuvuus tulevaisuuden ratkaisuihin

Kaikki kolme toteutettua algoritmia soveltuvat suorituskyvyltään varsin hyvin yleiskäyttöisiksi algoritmeiksi tulevaisuuden eri toteutuksiin. Kaikissa niissä voidaan suorituskykyä parantaa entisestään laitteistokohtaisia käskyjä hyödyntämällä. Tämän työn perusteella ei esteitä algoritmien soveltuvuudessa käyttöön eri kohteissa löytynyt.

Hilapohjaisten algoritmien implementointi on vaikeampaa kuin esimerkiksi RSA:n implementointi. Kuitenkin mitä tahansa kryptografista algoritmia toteutettaessa täytyy olla todella tarkkana, ja virheet ovat aina mahdollisia. Toteutuksen haastavuudesta johtuen voisi ajatella, että mahdollisia implementaatiossa piileskeleviä mokia on hankalampi paikallistaa. Kuitenkaan mielestäni ei sen pitäisi olla syy jättää ottamatta hilapohjaisia algoritmeja käyttöön. Virheitä on helppo välttää käyttämällä valmiita asiansa tuntevien ihmisten tekemiä toteutuksia sen sijaan, että toteuttaisi algoritmit itse. Loppujen lopuksi aina on olemassa riski mitä tahansa toteutusta käytettäessä, että siihen on jäänyt bugeja. Vaikka Dilithium ja Kyber ovat varsin monimutkaisia algoritmeja, eivät ne silti ole hila-algoritmeina monimutkaisimmasta päästä - NIST:n standardointikilpailun kolmannen kierroksen algoritmeista esimerkiksi FALCON on vielä monimutkaisempi toteuttaa kuin Dilitihum [40].

Kaikista tärkeintä algoritmien käyttöönotossa on se, että luotetaanko algoritmien turvallisuuteen. Monet hila-algoritmit perustuvat varsin tuoreisiin ongelmiin ja

siten niiden turvallisuuteen ja pitävyyteen ei välttämättä kannata luottaa niin lujasti kuin vanhempien algoritmien turvallisuuteen. Tämän hetken tietojen mukaan hilapohjaiset algoritmit ovat turvallisia sekä perinteistä että kvanttitietokonetta vastaan. Algoritmeista ja ongelmista, johon ne perustuvat, voi kuitenkin tulevaisuudessa löytyä heikkouksia. Hilapohjaisia algoritmeja vastaan lötyy erilaisia hyökkäyksiä, jotka rajoittavat muun muassa sitä, millainen salausjärjestelmä voi rakenteeltaan olla. Mikäli algoritmien turvallisuuteen ei luoteta, ei niitä oteta myöskään käyttöön. On monia järjestelmiä, joiden uudistaminen on hidasta ja kallista. Näihin järjestelmiin ei varmasti ensimmäisenä olla laittamassa hilapohjaisia algoritmeja.

Todennäköistä on, että standardoinnin jälkeenkin kvanttiturvallisten algoritmien käyttöönotto tapahtuu varsin verkkaisessa tahdissa. Koska kvanttitietokoneen aiheuttama uhka ei vielä juuri tällä hetkellä ole relevantti järjestelmien suunnittelussa, RSA:ta ja elliptisten käyrien salausmenetelmiä käytetään varmasti vastakin. Ensimmäisenä kvanttiturvallisia algoritmeja todennäköisesti otetaan käyttöön pienemmän mittakaavan nopeasti muokattavissa olevissa toteutuksissa. Mahdollisesti joissakin toteutuksissa siirrytään hybridimalliin avaintenvaihdossa, eli sekä perinteisten algoritmien että valinnan mukaan kvanttiturvallisten algoritmien käyttö on mahdollista. Joka tapauksessa kvanttitietokoneen kryptografisille algoritmeille aiheuttamaan uhkaan on syytä varautua. Valitettavasti on olemassa tunnettuja ennakkotapauksia siitä, että jo kauan sitten turvattomaksi todettuja algoritmeja on käytetty pitkälle eteenpäin, ja se on myöhemmin aiheuttanut isoja ongelmia.

7.5. Jatkoa työlle

Tässä työssä integroituja algoritmeja voi käyttää esimerkiksi erilaisissa demototetutuksissa. Niitä voi myös käyttää algoritmien turvallisuuden tai erilaisten hyökkäysten tutkimiseen. Itse toteutuksia itsessään voi myös tutkia, esimerkiksi erilaisilla dynaamisen ohjelma-analysin keinoilla tai vaikkapa fuzzaamalla [58]. Algoritmien käytännön hyödyllisyyttä voi tutkia myös korvaamalla niillä perinteisiä algoritmeja erilaisissa protokollissa tai muissa toteutuksissa. Tässä työssä tehdyn toteutuksen käytettävyyttä voisi tutkia myös käyttämällä niitä uusissa projekteissa perinteisten algoritmien sijaan. Tätä kautta voisi tutkia sekä toteutuksen käytettävyyttä ohjelmoijan ja ylläpitäjän (sekä mahdollisesti käyttäjän) näkökulmasta sekä algoritmien soveltuvuutta erilaisiin toteutuksiin.

Mikäli totetutuksia haluaisi liittää varsinaisen Crypto++-kirjaston yhteyteen, edessä olisi yhteyden ottaminen Crypto++:n ylläpitäjiin ja seuraavista askeleista sopiminen. Todennäköistä on, että järkevintä olisi odottaa NIST:n standardointikilpailun loppuun asti, sillä algoritmeihin voi tulla vielä muutoksia kolmannenkin kierroksen aikana. Sen jälkeen, mikäli jokin toteutetuista algoritmeista olisi valittu standardiksi, sen liittäminen kirjaston yhteyteen olisi ajankohtaisempaa.

Dilithium olisi hyvä päivittää ajankohtaiselle tasolle, sillä se on ehtinyt jo selvästi muuttua toteutetusta versiosta. Samaten muiden algoritmien toteutuksia olisi hyvä päivittää, mikäli niiden parametrejä tai muuta sisäistä elämää päivitetään standardointiprosessin aikana. Erinomaista olisi, mikäli joku asiantuntevampi henkilö, tai miksei useampikin, kävisi myös toteutukset läpi ja parantaisi ja optimoisi niitä.

Avoimeksi jää myös kysymys siitä, miksi SABERin suorituskyky tässä työssä toteutetussa versiossa on niin paljon huonompi kuin mallitoteutuksessa verrattuna Kyberiin tai Dilithiumiin, vaikka siihen piti tässä työssä tehdä vähiten muutoksia. Tähän vastauksen löytäminen jäi tämän työn raamien ulkopuolelle.

Myös muita kvanttiturvallisia algoritmeja näiden kolmen lisäksi voisi toteuttaa, samaten näitä kolmea algoritmia voisi toteuttaa jollain muulla ohjelmointikielellä. Koska nyt allekirjoittaneella on kokemusta siitä, miten algoritmien liittäminen C++-kirjastoon voisi tapahtua, seuraavaksi voisi kohteena olla vaikka Java-kielinen kirjasto. Koska Java-kielisiä esimerkkitoteutuksia ei tietääkseni algoritmeista ole, tämä tarjoaisi uudenlaisen lisähaasteen toteutustyöhön.

8. YHTEENVETO

Julkisen avaimen salausmenetelmiä käytetään esimerkiksi tietoliikenteen turvaamiseen erilaisissa kommunikaatioprotokollissa, kuten TLS tai SSH, ja digitaalisten allekirjoitusten tekemiseen. Digitaalisten allekirjoitusten avulla voidaan varmistaa, että dokumentin tai tiedoston kirjoittaja on sama kuin allekirjoittaja eikä sitä ole myöhemmin muokattu.

Julkisen avaimen salausalgorimit perustuvat vaikeisiin matemaattisiin ongelmiin. Kvanttitietokoneen tulo kuitenkin uhkaa näiden algoritmien turvallisuutta. Shorin algoritmi mahdollistaa tekijöihin jaon ja diskreettien logaritmien ongelmien ratkaisun merkittävästi nopeammin kvanttitietokoneen avulla kuin mitä perinteisellä tietokoneella on mahdollista. Niinpä esimerkiksi yleisesti käytettyjen RSA:n ja elliptisten käyrien salausmenetelmien turvallisuus romahtaa, mikäli riittävän tehokkaita kvanttitietokoneita on tulevaisuudessa saatavilla.

Tätä varten on kehitetty eri ongelmiin perustuvia algoritmeja, joiden tarkoitus on olla kvanttiturvallisia eli niiden turvallisuus ei merkittävästi heikenny, mikäli hyökkääjällä on käsissään tehokas kvanttitietokone. Hiloihn pohjautuvat algoritmit ovat yksi tällainen luokka. Tässä työssä integroitiin Crypto++-ohjelmistokirjaston kopioon kolme NIST:n käynnissä olevan kvanttiturvallisten algoritmien standardointiprosessin finalistialgoritmia. Integroiduista algoritmeista kaikki perustuvat hilarakenteisiin. Kyber ja SABER ovat tarkoitettu käytettäviksi erityisesti avaintenvaihdossa, kun taas Dilithium digitaalisissa allekirjoituksissa.

Työn tuloksena kaikki kolme algoritmia liitettiin kirjaston kopion yhteyteen ja niiden suorituskykyä sekä turvallisuutta analysoitiin. Suorituskyky oli suurimmaksi osaksi hieman heikompi kuin mallitoteutuksissa, joita käytettiin perustana tässä työssä. Turvallisuusongelmia ei löytynyt, mutta erityisen tarkkaa turvallisuusanalyysiä toteutuksesta ei tämän työn puitteissa ollut mahdollista tehdä.

9. VIITTEET

[1] Rivest R.L., Shamir A. & Adleman L. (1978) A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM 21, s. 120–126.

[2] Hankerson D., Menezes A.J. & Vanstone S. (2006) Guide to elliptic curve cryptography. Springer Science & Business Media.

[3] Rescorla E. (2018), The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446. URL: https://rfc-editor.org/rfc/rfc8446.txt.

[4] Lonvick C.M. & Ylonen T. (2006), The Secure Shell (SSH) Authentication Protocol. RFC 4252. URL: https://rfc-editor.org/rfc/rfc4252.txt.

[5] Shor P.W. (1997) Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM Journal on Computing 26, ss. 1484–1509.

[6] Proos J. & Zalka C. (2003) Shor's discrete logarithm quantum algorithm for elliptic curves. Quantum Information & Computation 3, ss. 317–344.

[7] OpenSSL. URL: https://openssl.org/. Vierailtu 8.4.2021.

[8] Crypto++ Library 8.5|Free C++ Class Library of Cryptographic Schemes. URL: https://cryptopp.com/. Vierailtu 9.3.2021.

[9] Tässä työssä käytetty ohjelmistokirjaston kopio. URL: https://github.com/juliushekkala/cryptopp-pqc. Vierailtu 26.5.2021.

[10] Shoup V. (2001), A Proposal for an ISO Standard for Public Key Encryption. Cryptology ePrint Archive, Report 2001/112. URL: https://eprint.iacr.org/2001/112.

[11] Aumasson J.P. (2017) Serious Cryptography: A Practical Introduction to Modern Encryption. No Starch Press, 312 s.

[12] Diffie W. & Hellman M.E. (1976) New directions in cryptography. IEEE Transactions on Information Theory 22, ss. 644–654.

[13] Rescorla E. (2000), HTTP Over TLS. RFC 2818. URL: https://rfc-editor.org/rfc/rfc2818.txt.

[14] McCurley K.S. (1990) The discrete logarithm problem. Teoksessa: Proc. of Symp. in Applied Math, nide 42, USA, nide 42, ss. 49–74.

[15] Miller V.S. (1986) Use of elliptic curves in cryptography. Teoksessa: H.C. Williams (toim.) Advances in Cryptology — CRYPTO '85 Proceedings, Springer Berlin Heidelberg, Berlin, Heidelberg, ss. 417–426.

[16] Koblitz N. (1987) Elliptic curve cryptosystems. Mathematics of computation 48, ss. 203–209.

[17] Barker E. (2020) Recommendation for Key Management: Part 1 – General. Tekninen raportti, NIST Special Publication 800-57 Part 1 Revision 5, National Institute of Standards and Technology.

[18] Bernstein D.J. (2006) Curve25519: new Diffie-Hellman speed records. Teoksessa: International Workshop on Public Key Cryptography, Springer, ss. 207–228.

[19] Johnson D., Menezes A. & Vanstone S. (2001) The elliptic curve digital signature algorithm (ECDSA). International journal of information security 1, ss. 36–63.

[20] Kaur R. & Kaur A. (2012) Digital signature. Teoksessa: 2012 International Conference on Computing Sciences, ss. 295–301.

[21] National Institute of Standards and Technology (2013) Digital Signature Standard (DSS) (FIPS PUBS 186-4). Standardi, U.S. Department of Commerce.

[22] DiVincenzo D.P. (1995) Quantum computation. Science 270, ss. 255–261.

[23] Steane A. (1998) Quantum computing. Reports on Progress in Physics 61, s. 117–173.

[24] Grover L.K. (1996) A fast quantum mechanical algorithm for database search. Teoksessa: Proceedings of the twenty-eighth annual ACM symposium on Theory of computing, ss. 212–219.

[25] Arute F., Arya K., Babbush R., Bacon D., Bardin J.C., Barends R., Biswas R., Boixo S., Brandao F.G., Buell D.A. et al. (2019) Quantum supremacy using a programmable superconducting processor. Nature 574, ss. 505–510.

[26] Bernstein D.J. (2009) Introduction to post-quantum cryptography. Teoksessa: Post-quantum cryptography, Springer, ss. 1–14.

[27] Post-Quantum Cryptography | CSRC. URL: `https://csrc.nist.gov/projects/post-quantum-cryptography/`. Vierailtu 23.9.2020.

[28] Crockett C., Paquin C. & Stebila D. (2019) Prototyping post-quantum and hybrid key exchange and authentication in TLS and SSH. Tekninen raportti, Open Quantum Safe.

[29] Grätzer G. (2011) Lattice theory: foundation. Springer Science & Business Media.

[30] Olaussen O. (2017) Hilat ja kryptografia. Pro gradu -tutkielma, Oulun yliopisto, matemaattisten tieteiden laitos.

[31] Regev O. (2005) On lattices, learning with errors, random linear codes, and cryptography. Teoksessa: Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing, STOC '05, Association for Computing Machinery, New York, NY, USA, s. 84–93.

[32] Bos J., Ducas L., Kiltz E., Lepoint T., Lyubashevsky V., Schanck J.M., Schwabe P., Seiler G. & Stehlé D. (2018) CRYSTALS-Kyber: a CCA-secure module-lattice-based KEM. Teoksessa: 2018 IEEE European Symposium on Security and Privacy (EuroS&P), IEEE, ss. 353–367.

[33] Ducas L., Lepoint T., Lyubashevsky V., Schwabe P., Seiler G. & Stehlé D. (2018) Crystals–dilithium: Digital signatures from module lattices .

[34] D'Anvers J.P., Karmakar A., Roy S.S. & Vercauteren F. (2018) Saber: Module-LWR based key exchange, CPA-secure encryption and CCA-secure KEM. Teoksessa: International Conference on Cryptology in Africa, Springer, ss. 282–305.

[35] Hoffstein J., Pipher J. & Silverman J.H. (1998) NTRU: A ring-based public key cryptosystem. Teoksessa: International Algorithmic Number Theory Symposium, Springer, ss. 267–288.

[36] Banerjee A., Peikert C. & Rosen A. (2012) Pseudorandom functions and lattices. Teoksessa: D. Pointcheval & T. Johansson (toim.) Advances in Cryptology – EUROCRYPT 2012, Springer Berlin Heidelberg, Berlin, Heidelberg, ss. 719–737.

[37] Falcon: Fast-Fourier Lattice-based Compact Signatures over NTRU. URL: https://falcon-sign.info/. Vierailtu 5.3.2021.

[38] McEliece R. (1978) A Public-Key Cryptosystem Based On Algebraic Coding Theory. DSN Progress Report 44, ss. 114–116.

[39] Bernstein D.J. & Lange T. (2017), Post-quantum cryptography—dealing with the fallout of physics success. Cryptology ePrint Archive, Report 2017/314. URL: https://eprint.iacr.org/2017/314.

[40] Alagic G., Alperin-Sheriff J., Apon D., Cooper D., Dang Q., Kelsey J., Liu Y.K., Miller C., Moody D., Peralta R. et al. (2020) Status report on the second round of the NIST post-quantum cryptography standardization process. Raportti, National Institute of Standards and Technology.

[41] Ding J. & Schmidt D. (2005) Rainbow, a new multivariable polynomial signature scheme. Teoksessa: J. Ioannidis, A. Keromytis & M. Yung (toim.) Applied Cryptography and Network Security, Springer Berlin Heidelberg, Berlin, Heidelberg, ss. 164–175.

[42] Ding J. & Yang B.Y. (2009) Multivariate public key cryptography. Teoksessa: Post-quantum cryptography, Springer, ss. 193–241.

[43] Merkle R.C. (1989) A certified digital signature. Teoksessa: Conference on the Theory and Application of Cryptology, Springer, ss. 218–238.

[44] Heartbleed bug. URL: https://heartbleed.com/. Vierailtu 9.3.2021.

[45] Cve-2020-0601. URL: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-0601. Vierailtu 9.3.2021.

[46] Lazar D., Chen H., Wang X. & Zeldovich N. (2014) Why does cryptographic software fail? A case study and open problems. Teoksessa: Proceedings of 5th Asia-Pacific Workshop on Systems, ss. 1–7.

[47] Egele M., Brumley D., Fratantonio Y. & Kruegel C. (2013) An Empirical Study of Cryptographic Misuse in Android Applications. Teoksessa: Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security, CCS '13, Association for Computing Machinery, New York, NY, USA, s. 73–84.

[48] Gaj K. (2018) Challenges and rewards of implementing and benchmarking post-quantum cryptography in hardware. Teoksessa: Proceedings of the 2018 on Great Lakes Symposium on VLSI, GLSVLSI '18, Association for Computing Machinery, New York, NY, USA, s. 359–364.

[49] Open Quantum Safe. URL: https://openquantumsafe.org/. Vierailtu 11.3.2021.

[50] liboqs: C library for prototyping and experimenting with quantum-resistant cryptography. URL: https://github.com/open-quantum-safe/liboqs/. Vierailtu 11.3.2021.

[51] Post-quantum cryptography | round 3 submissions. URL: https://csrc.nist.gov/Projects/post-quantum-cryptography/round-3-submissions. Vierailtu 14.4.2021.

[52] Avanzi R., Bos J., Ducas L., Kiltz E., Lepoint T., Lyubashevsky V., Schanck J.M., Schwabe P., Seiler G. & Stehlé D., CRYSTALS-Kyber — algorithm specifications and supporting documentation (version 3.01). URL: https://pq-crystals.org/kyber/data/kyber-specification-round3-20210131.pdf. Vierailtu 7.6.2021.

[53] National Institute of Standards and Technology (2016), Submission requirements and evaluation criteria for the post-quantum cryptography standardization process. URL: https://csrc.nist.gov/csrc/media/projects/post-quantum-cryptography/documents/call-for-proposals-final-dec-2016.pdf. Vierailtu 7.6.2021.

[54] Fujisaki E. & Okamoto T. (1999) Secure integration of asymmetric and symmetric encryption schemes. Teoksessa: Annual International Cryptology Conference, Springer, ss. 537–554.

[55] Templates - cppreference.com. URL: https://en.cppreference.com/w/cpp/language/templates. Vierailtu 12.3.2021.

[56] GCC, the GNU Compiler Collection. URL: https://gcc.gnu.org/. Vierailtu 20.5.2021.

[57] Valgrind. URL: https://valgrind.org/. Vierailtu 20.5.2021.

[58] Vranken G. (2019), Differential fuzzing of cryptographic libraries. URL: https://guidovranken.com/2019/05/14/differential-fuzzing-of-cryptographic-libraries/. Vierailtu 5.5.2021.